ABSTRACT

SACHIDANANDA, SUBASH GHATTADAHALLI. Adding Rivalrous Hardware Scheduling to the First Generation FREEDM Systems Communication Platform. (Under the direction of Dr. Alexander G. Dean.)

Existing power management systems are hard-wired, slow, unreliable and insecure. By improving the communication framework for power management systems, using Internet and modern communication protocols like IEC61850, Zigbee (IEEE 802.15.4), etc., not only can energy be managed better, but also, the system as a whole can be made faster, secure and reliable. Any communication network has certain important characteristics like delay, range, scalability, network topology, etc., that dictate its effectiveness and usefulness in a particular environment. Delays in various parts of the network is one of the important characteristics that needs to be studied carefully. Network delays can be reduced by not only faster hardware but also better software. Importance also needs to be placed on monitoring power consumption of the devices, and ways to improve power efficiency of the system.

In conjunction with that, a first generation communication framework for renewable energy distribution and management system, was set up using a network of embedded boards and personal computers. Various communication protocols and interfaces were tried to prove the versatility and reliability of the entire system. Experiments were conducted to test the range and delays in various parts of the communication network. Having established a platform for the nodes to communicate, investigation was done to implement techniques that could make the nodes more power efficient. One of the ways to stretch the battery life is through addition of an SMPS. However, addition of an SMPS introduces power instability to the system and interferes with normal functioning of sensitive devices like ADC, compass, etc. A processor controlled SMPS was added to the communication platform and its interference with normal functioning of the system was studied. Possible hardware and software approaches to counter this interference are also presented.

Adding Rivalrous Hardware Scheduling to the First Generation FREEDM Systems Communication Platform

by

Subash Ghattadahalli Sachidananda

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Master of Science

Computer Engineering

Raleigh, North Carolina

2010

APPROVED BY:

Dr. Eric Rotenberg	Dr. Frank Mueller

Dr. Alexander G. Dean
Chair of Advisory Committee

DEDICATION

To my parents, Smt. Manjula and Shri G. R. Sachidananda

BIOGRAPHY

After leading a relatively easy life for the first twenty three and half years in Bangalore, India, Subash realized that, he needed a change in life. He felt it was time to be on his own, experience life in a different place, and most importantly feed his brain with some much needed creative and intellectual thoughts. Pursuing a Master degree was the noblest path he could find to do all the above.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Alexander Dean, for giving me an opportunity to work with him, and for spilling light whenever I have lost the path in darkness. The encouragement and support he has given me as an advisor, as a professor and as a friend, has been immense and immeasurable. I would like to thank Dr. Mueller and Dr. Rotenberg for being on my committee, and for their valuable feedback. I thank Dr. Wenye Wang for her guidance and support. Her suggestions were very helpful while working on the RSC project. Studying at North Carolina State University has been challenging, and the atmosphere has been of high quality. I am thankful for getting an opportunity to be part of such a community. I thank Mohit, who worked with me in the RSC project, Karthik S., who helped me set up the MSP-GNU tool chain, and Greg Parsons, for his help in the RHS project. I also like to thank my lab mates, Sangyeol Kang and Shaolin, for their support. Back home, I thank Mr. Uday Nandiwada for, his guidance and, going out of his way to help me pursue a Master degree, without which, this would not have been possible. I thank Dr. H. G. Nagendra and Dr. Rama Murthy for their encouragement. I would like to thank my parents, who have been the pillars of unconditional love, care and support all my life, and have stood by my decisions all along. They have always given me freedom to pursue my dreams. Lastly, I thank my close friends, with whom, I have shared and cherished some of those unforgettable moments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 About FREEDM systems center and existing energy management systems

Production, management and delivery of electrical energy has always been a topic of great interest for thinkers around the world. With depleting fossil fuels and the cost of producing conventional energy being high, emphasis has been laid on exploring long term sustainable energy sources. These primarily refer to renewable energy sources like solar, wind, geo-thermal, tidal, etc. which not only satisfy energy requirements but also do so in an environmental friendly manner. Most of the renewable energy is generated and stored in remote locations and transmission of such stored energy is to be done with minimal losses to ensure efficiency of the total system. The same applies to conventional energy generation. Thus energy delivery and management plays a pivotal role in power generation and problems in these areas need to be addressed with prompt attention.

The Future Renewable Electric Energy Delivery and Management (FREEDM) systems center [1] has been set up by the National Science Foundation (NSF) in collaboration with a number of companies and universities, including NCSU, to emphasize on smart energy generation, management and delivery. One of the goals of the center is to establish distributed renewable energy sources. Having said that, establishing renewable energy resources does not alone solve the entire problem. There is also a need for an infrastructure to deliver and manage these resources more efficiently. This shifts the focus from centralized energy generation and storage to doing so at various places like residential areas, commercial customer

locations, etc. and distribution of excess energy generated, to other places with higher power requirements or even back to the power generation units. One of the prime goals of FREEDM is to establish smart grids, which supports, plug and play of energy storage resources, energy management through distributed grid intelligence, and a scalable, reliable and secure communication backbone.

A typical energy management system monitors the load requirements for a generation unit and adjusts the energy generated accordingly. In some cases, excess energy produced in one grid is transferred to another, with higher requirements, so that the system as a whole maintains a balance between production and consumption. The existing energy management systems [2] are hardwired and unreliable. They are error prone and do not support fault recovery and hence faults in the monitoring system may cause unrecoverable damage to both power generation systems and the consumers. There is also human interaction for monitoring and control which induces another dimension of unreliability.

1.2 Need for a Reliable and Secure Communication (RSC) platform

With the inception of the Microgrid [3] concept, stress has been laid on building autonomous power generation units that cater to a small scale network of power hungry units like floors of a big building or a small group of houses in an area or an island which require not more than 100kW. Not only do these grids generate their own power, but they can also coexist and draw power from conventional power lines if required. They are even designed to transfer excess energy generated within them to other grids that have higher power requirements. In case of line faults, these microgrids are designed to isolate themselves from the main grid so that their functioning is not impacted.

All the above qualities of a microgrid rely on a strong feedback sensor and control mechanism, that monitors generator capacity, reports power consumption, provides fault detection and controls functioning of the microgrid based on all several input parameters. Existing microgrids use control systems that have hard real-time requirements. For example, a technical report on ‘ Intelligent Load Control’ [4] cites that disturbances like power shortage can cause a frequency dip in the supply and a dip of 0.8 Hz has to be detected and addressed within 7 to 15 seconds and the relevant micro-sources in the grid must be turned on to

compensate for the power shortage and bring up the frequency to normal. If the sources are already running at full steam, some of the non-critical loads need to be shed until normalcy is restored.

In order to implement such features, a reliable and secure communication platform is required. Many existing feedback control loops are unreliable and not robust enough. Considering this, FREEDM systems require a new kind of communication platform that not just meets real-time requirements, but is also flexible to the maximum extent, redundant, secure and able to support plug and play of different types of hardware and software devices and interfaces. Hence, one of the prime goals for FREEDM center is to establish a reliable, scalable and secure communication (RSC) platform, using various modern communication protocols and interfaces, which incidentally forms the backbone of energy management and distribution. This RSC platform acts as a base by supporting communication among a plethora of systems such as Intelligent Energy Management systems(IEM), Intelligent Fault Management (IFM) systems, Solid State Transformers (SST), Distributed Grid Intelligence system (DGI), etc. All these systems put together form an integral part of the future energy delivery and management system.

1.3 Rivalrous Hardware Scheduling (RHS) on FREEDM systems

As cited by [4], monitoring systems in power management need to satisfy hard real-time deadlines. In a DGI system, for example, nodes use RSC system to constantly interact with each other and form an intelligent network so that it helps a node realize when another node in its critical path goes down and quickly approach a new node to maintain its connectivity with the network. Such dependencies make the communication backbone for FREEDM systems a hard real-time (HRT) system. Hence, for the RSC set up, the communication delay has to be maintained within a strict threshold. Along with checking delays, the network nodes need to be made as power efficient and cost effective as possible. Rivalrous Hardware Scheduling (RHS) is a scheduling policy which interleaves aspects of real-time scheduling with the aim of minimizing underlying hardware cost and maximizing power efficiency. RHS works on the basic idea that, some rivalrous hardware components

need to run and stop only at specific points in time and by carefully scheduling their run times, interference among them can be avoided, thus reducing the need for using cost and area consuming noise filters. This makes the system more power efficient since sources like Switch Mode Power Supply (SMPS) can now be added to the system and at the same time provides room for cost cutting by removing the need for expensive noise filters without disturbing the normal functioning of existing system.
Chapter 2 describes the hardware and software platform used to set up a first generation platform for RSC. Chapter 3 and 4 present the implementation of the platform and experiments run on the same. Chapters 5 , 6 and 7 explain the RHS model and its implementation on the RSC platform while Chapter 8 describes results of experiments run on the system with RHS. Chapter 9 presents conclusion and future work.

1.4 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 [5]. However, they are not effective against high frequency noise. Articles [6] and [7] discuss about 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 [7], shielding is utilized to reduce magnetic interference around choke coils. [7] 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. [8] and [9] 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 [10] 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 [10] does not suffer from EMI and hence hardware scheduling in [10] is not

done with the primitive purpose of reducing EMI.

Chapter 2

Environment used for RSC platform

2.1 Two tier architecture for FREEDM systems

The prototype communication backbone for FREEDM systems needs to be reliable, robust and scalable enough to be able to support a variety of protocols related to power systems (like IEC61850 [11], DNP3 [12]), communication (like TCP/IP) and different applications. Apart from protocols, the prototype should also contain different communication interfaces like USB, Ethernet, UART, etc in order to connect a variety of systems together.

In the RSC system, even though at a logical level, all nodes are part of a unified network and can interact with one another, at the physical level, the underlying hardware, software, communication interfaces and their functionalities make them different. Hence, the communication backbone for FREEDM was set up as a two tier architecture - the top level, containing nodes running at higher speeds, supporting larger packet sizes of kilo or mega bytes, covering a wider area and a bottom level, containing nodes running at lower speeds, supporting smaller size packets of a few hundred or kilo bytes and covering a smaller area. The top level network would be used for communication among IFMs, IEMs, generating station, etc. that are spread out over a wide area, spanning several miles. The bottom level network would be used for passing around sensor data, like temperature, supply voltage, etc. and control packets between nodes within a big room or a single floor in a building.

Such a low level would contain a special node that, on one hand, would hook up with several peer nodes in the lower level and on the other hand, interface with one of the nodes at the top level, thus linking up both tiers. In order to set up this two tier architecture, two different types of embedded boards were used.

2.2 Underlying hardware for the communication tiers

The top level system was set up using ARM based 32-bit micro-controller boards from Technologic Systems called 'TS7250' [13]. The board is run by embedded Linux at 200 MHz with 64MB of SDRAM and 128MB of NAND Flash memory. The board also supports 10/100 Ethernet port, USB 2.0, two high speed serial ports and a PC/104 expansion bus. High speed, big memory, support for various interfaces and many more features make it an ideal board for nodes in the top tier of the network model.
The low level network was set up using MSP430 based 16 bit low power micro-controller nodes from Texas Instruments called 'eZ430-RF2480' [14]. These nodes work at a wide range of supply voltage (1.8 to 3.6 V) and frequency (1 MHz to 16 MHz) and contain UART and SPI interfaces for communication. These boards also contain a CC2480 radio chip which has all the supporting hardware and software built-in to sustain wireless Zigbee (IEEE 802.15.4) [15] communication protocol, using which the application code on the MSP (short for MSP430), communicates and shares data. These boards can be run by a battery pack or by a DC power supply. Such boards are ideal in setting up a small personal area network of a sensor nodes that sense signals and send relevant data packets to their respective parent nodes.

2.3 Test setup for RSC platform

The figure 2.1 shows the test bed set up to implement first generation platform of RSC for FREEDM systems. Two 7250 boards are deployed at *CESR* [1] and *NETWIS* [2] labs. It can be seen from the figure that several interfaces such as LAN, Wireless LAN, Zigbee

[1] CESR: Center for Efficient, Scalable and Reliable Computing at Partners I building, NCSU.
[2] NETWIS: Networking of Wireless Information Systems Lab at Engineering Building II, NCSU.

and Serial port communication have been tried and tested to demonstrate the scalability and flexibility of the communication platform.

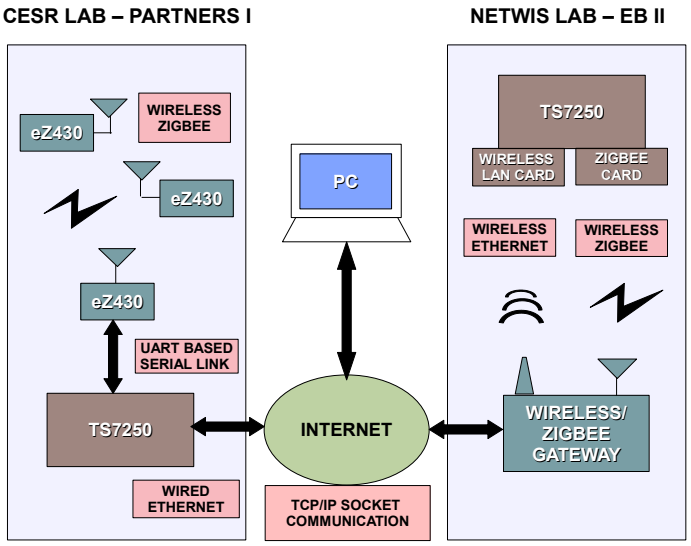


Figure 2.1: The various communication paths and interfaces set up as part of RSC backbone

As shown in the figure, the TS7250, in the *NETWIS* lab, was interfaced with a wireless LAN card, a zigbee card and wired LAN to establish connection with the Internet. On the other hand, the TS7250 at the *CESR* lab, connects to the Internet over wired LAN and to the low tier Zigbee network of eZ430 nodes over UART serial interface.
The figure also shows a personal network of three eZ430 sensor nodes in *CESR* lab, communicating over the Zigbee interface. The node that interfaces with the TS7250 is called the *Coordinator* and is the node that comes up first and starts the personal Zigbee network. Out of the remaining two eZ430 nodes, one is a *Router* and the other an *End device* [3]. The router and the end devices can be collectively addressed as *Source* nodes since they both sense and send data to the coordinator. However, the difference being, a router forwards packets from an end node that cannot access the coordinator directly. This means that any router node is always in contact with the coordinator. The only function of the end node is to send the data it senses periodically to the coordinator, either directly or through a

[3]Please look up the Zigbee specification document available at [14] for more details on the Zigbee node types and other role specific functionalities

router. Once the coordinator sets up the network, the router and end nodes can join in and start sending the data they sense.

Chapter 3

Implementation of the communication backbone for RSC

3.1 Linking up different devices

3.1.1 Top tier of RSC

Under current implementation, the TS7250 nodes in the top tier of RSC, interact with each other over TCP/IP based socket communication. Since the boards are run by embedded Linux, the operating system provided the underlying lower layered TCP/IP protocol support. However, a layer of the application code was developed for RSC, which sits on top of the OS and handles system calls for socket based communication. This socket code is separate from the main communication code and it only provides hooks for socket creation and management. It can be easily detached, replaced or changed if a different platform or low layer communication protocol is used in future, without much change to the communication code and this adds flexibility to the software.

With the use of socket based communication, not only do the TS7250 boards interact with each other but also connect with a PC or other devices that support TCP/IP, as shown earlier in figure 2.1. The figure also depicts work done in parallel at the *NETWIS* lab, where in the TS7250 talks to a gateway via wireless LAN and Zigbee cards, connects to the Internet and eventually to the other TS7250 board in the *CESR* lab. This demonstrates the adaptability of the RSC platform. The wireless interfaces ensure that the TS7250 nodes

connect to the Internet even from places not supporting a LAN connection. This also means that, in case one of the interfaces is faulty, communication can still happen using alternate routes and this makes the system robust.

3.1.2 Bottom tier of RSC

Most of the work done at the *CESR* lab for RSC, involves setting up two-way communication paths among devices in the bottom tier and interfacing them to nodes in the top tier. The eZ430 nodes used for setting up the bottom tier, communicate with each other over Zigbee. The characteristics of Zigbee used by the eZ430 nodes is different from that used by TS7250 nodes, mentioned earlier, and even though they support comparatively lower bandwidth and data rates, they draw less power and hence more energy efficient. A sample code provided by *Texas Instruments*, called Zigbee Acceleration Sample Application (ZASA) [14], was used as the base code for communication in this tier. This sample code comprises of an application that configures the node to set up a personal Zigbee network and defines the role of a particular node [1] in the network depending on certain parameters. However this sample code only supported one-way communication - from the routers or source nodes to the coordinator and hence additional code was added to support two-way communication, along with other features to make it suitable for RSC.

The ZASA code runs on the MSP430 processor and accesses the Zigbee protocol stack available in the CC2480 chip, over SPI, in order to perform wireless communication. This functionality was retained since it formed a core aspect of communication for these nodes. The original circular buffer implementation to send and receive data using the UART module was also used in interfacing these nodes with the UART of the TS7250 board. The UART on the TS7250, however is handled by the OS, which provides a pseudo-file for the device. A UART layer for the communication code on the TS7250 was developed anew and it sits between the OS and the application code and uses this pseudo-file to send and receive data over the UART. Like the socket code, the UART code provides hooks to create file connections and to do data transfers, by hiding the OS specific calls from the main communication code. This way, a change in the platform and/or OS need not inflict a change in the actual communication code and such an approach would be useful going forward into the

[1] The various roles of the nodes in the network were briefly mentioned in 2

future generations of RSC, where in complex power management protocols like IEC61850, DNP3, etc would be added.
An addition to the source code, mentioned earlier, running on the MSP430 nodes, was done for establishing a communication path from the coordinator to the routers and source nodes. This completed the final two-way communication system between the two tiers and among nodes within the same tier. This way, the TS7250 in the *NETWIS* lab can send information to an MSP430 based zigbee node in the *CESR* lab and vice-versa using several communication means as shown in figure 2.1 of chapter 2.

3.2 Types of data used by RSC

As mentioned earlier in chapter 2, the two-tier architecture of RSC can be viewed as one single logical network of nodes, linked up using several common interfaces, even though they use different platforms. Such a logical network of nodes would have the need to share data and control information among nodes, irrespective of the physical tier they belong to.
In the current implementation, the Zigbee source nodes send room temperature and supply voltage of the board to the coordinator. The software running on eZ430 boards uses dedicated channels within the Analog-to-Digital Conversion (ADC) module of the MSP430 to measure temperature and on board supply voltage. The code also configures a timer, with a user settable timeout period (say 5 seconds), present within the application code itself, to repeatedly send these two values to the coordinator at that set period. The nodes place themselves in sleep mode, when not sending data and hence consume less power. In order to differentiate data sent by the various nodes, the application code also attaches 16-bit addresses that uniquely identify each node. The address format is explained in more detail in the next section. The coordinator collects data sent by the router/source nodes and relays the same to the TS7250 over the UART. The maximum baud-rate common to both devices is 38.4 KBaud, and hence that is the maximum baud-rate at which the communication takes place between the lower and the upper tiers. This restriction is caused by the eZ430 nodes, whereas the TS7250 can support higher baud-rates of upto 115.2 KBaud. Once the data reaches the TS7250 it is stored locally in different files based on the address of the node that sent it. The stored local data is read by a web based monitoring tool, on

a remote PC, by invoking an SNMP [16] daemon running on the TS7250, and is stored in a centralized database. Data is picked up from the centralized data base and mapped onto a website periodically so that it can be drawn into graphs and charts for analysis.
The TS7250 in *NETWIS* can also communicate with any of the eZ430 devices in the *CESR* lab by using the 16-bit logical address. This 16-bit address can be changed by the application on-the-fly [2]. At startup, the TS7250 at *NETWIS* sends the report interval with which each node should send temperature and voltage readings periodically. Such a configuration command first traverses through the gateway onto the Internet and reaches the TS7250 at *CESR*. This in turn forwards the data to the coordinator over the UART and the coordinator broadcasts it across to all its child nodes over Zigbee. All child nodes read the data, but the one being addressed to, responds by changing its report period. The other nodes discard the packet. This completes the two way communication among the nodes. The broadcast from the coordinator to the source nodes could be both advantageous and disadvantageous. It could be useful to send data common to all the nodes at once rather than repeatedly and thus saves time. But in case there is data that is specific to a node, other nodes would unnecessarily absorb and reject the data. Investigating such details and network topology is part of future work.

3.3 Unified packet format for communication

3.3.1 Requirements for the packet

Packets form an essential bond that holds a network together. In order to support communication between the two tier platform, a unified packet format had to be created so that the systems at both levels could share data among one another. Since this was a first generation communication platform, simple protocols and fewer nodes were used. The packet format used for such a communication network had to satisfy several requirements:

1. It had to be flexible enough to accommodate data of different formats and sizes.

2. The packet had to be scalable enough to support communication among a large number of nodes in future.

[2]The Zigbee stack on eZ430 also support 16 bit and 64 bit IEEE addresses, but using these addresses would hard code the nodes with specific addresses, in the non-volatile memory and limit the flexibility of the nodes

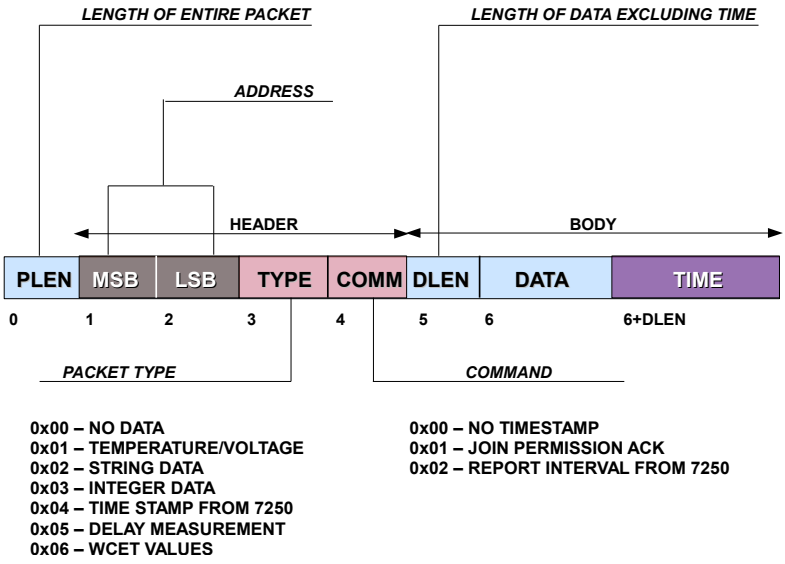


Figure 3.1: Packet format used for first generation RSC platform

3. It had to be platform or hardware independent so that it could be used by nodes at both levels.

4. The packet also had to be independent of underlying protocol and interfaces, in case they change in future.

3.3.2 Format of the packet

Based on the above factors, the packet format created for RSC is shown in figure 3.1. The packet contains a 4-byte header section and a variable length body section.

Address

The header section contains space for storing 16 bit addresses, which uniquely identify nodes in the network at the application layer. This way, each node, irrespective

of the level or the tier it belongs to, would have a 16 bit logical address and be part of a unified logical network. As of current implementation, a packet flowing from a lower level to a higher level would have the address of the node sending the packet and that coming from the top level would have the address of the node at the lower level, to which the packet has to be sent. Addresses *0X0000* and *0XFFFF* are currently reserved for either broadcast or similar use. This corresponds to the fact that a personal network of more than 65000 nodes in either tier can simultaneously talk to one another using this packet. The address field can be expanded, may be to three or four bytes, to accommodate more nodes in future.

Packet Type and Command

The *Packet Type* and the *Command* work in conjunction to define the contents of the packet in a better way, to help the receiver nodes. They complete the header section along with the *Address* field. The *Type* gives the type of data present in the packet, if any, and the *Command* field is used by nodes to send possible actions that needs to be taken by the other upon receiving the packet. The figure 3.1 also shows some of the values being used currently for the *Packet Type* and the *Command* fields. For example, a *Command* field of *0x02* would indicate a Zigbee source node to change its reporting interval to the new value present in the data section. The *Packet Type* in the above case would be *0x03* indicating the data is an integer. These two fields help make the packet flexible for sending not just different types of data but also commands and acknowledgments, based on which other nodes react and/or configure themselves.

Length

There are two length fields in the packet - *Packet length* and *Data length*. The *Packet length* is the size of the entire packet, inclusive of header and body. It is mainly used by the communication device driver software running on modules like UART, Zigbee, etc, at the time of sending/receiving packet. Since the *Packet length* occupies the first byte, the device driver can figure out how many bytes are coming in or going out as soon as it sees the first byte. The *Data length* on the other hand specifies the size of the relevant application data, excluding the size of the time stamp section. Since the data field itself is optional, the length would be set to *0* when the packet does not contain data. The corresponding

Packet Type would indicate the type of data, if present. As seen in figure 3.1 a value of *0x02* would indicate that the data field needs to be interpreted as characters and so on. A *Packet Type* of *0x00* would indicate absence of the data section.

Time stamps

The packet is also flexible enough to relay time stamps along with the data. Such a field is important for sensor nodes that send power related data. A time stamp in real-time, along with the data, would be very useful in either analyzing when fault occurred in a system or aggregating the data over a period of time and mapping the power consumption with respect to time of the day. The packet supports relay of time stamps in seconds and microseconds with up to $1\mu s$ resolution. Currently the size of the time field is *0x07* with the first 3 bytes for sending microseconds and the next 4 bytes for sending time in seconds.

3.4 Real-time clock setup and synchronization

The Linux kernel running on the TS7250 board initially supported time resolutions of 10 ms. This was not precise enough to either keep track of time or do delay measurements, since the speed of communication in the top tier is much higher. In order to counter this, High Resolution Timer patches, available on the Technologic System website, were downloaded and applied to the Linux kernel source code. This newly compiled kernel image was loaded onto the board and this improved the time resolution, from 10 ms to $1\mu s$.
In an effort to establish attachments of time stamps to the data, the timer on the MSP was configured to keep track of time with resolutions up to $1\mu s$. Since the eZ430 nodes cannot directly connect to the Internet to set local time, the TS7250 sends a packet to the eZ430 nodes with the initial real time. The TS7250 was also configured to send time stamps periodically, in order to synchronize the Timer on the MSP. However, the application code on the MSP stops the clock source for the timer when it goes to sleep mode. Hence the accuracy of the time on the eZ430 would be off by a few milliseconds. To avoid this situation, the Timer on the MSP can be run with a different clock source, which runs even during low power mode, but at the loss of time precision from $1\mu s$ to around $30\mu s$. The precision requirements of the timer depends on the final application and are unknown at present. Hence, establishing reliable time synchronization also forms part of the future work.

Chapter 4

Evaluation of first generation platform of RSC

4.1 Test for range

The TS7250 boards have several interfaces and contain built-in protocol suites to handle a variety of communication devices. With support for connecting to the Internet, these nodes can communicate over very large distances. The communication range of eZ430 Zigbee nodes, however, is limited since they are not very powerful. But they are very much part of the logical global network talked about earlier and can form a bottleneck in critical communication paths. Hence, the maximum physical distance between two eZ430 nodes, beyond which the Zigbee radio is unsuccessful to communicate, had to be determined. The physical range of the RF antenna on these nodes was tested both indoors and outdoors. The tests were conducted in and around the *CESR* lab at *Partners I* building at NCSU. The results are shown in table 4.1. The range is less outside compared to the range inside and could be accounted to higher signal loss and attenuation outdoors.

4.2 Test for delays in communication path

The UART interface between the coordinator eZ430 node and the TS7250 node forms the bottleneck between upper and lower tiers of the network. Also, the wireless Zigbee

Table 4.1: The maximum range of eZ430 nodes

	Parking lot (Outdoors)	**Hallway (Indoors)**
Distance in m	20.73	36.51

interface is an essential part of the low level communication system and delays within this could significantly slow down overall communication. Hence the critical path was identified to be the one from the TS7250 to the source nodes via the coordinator and back, as shown in figure 4.1.

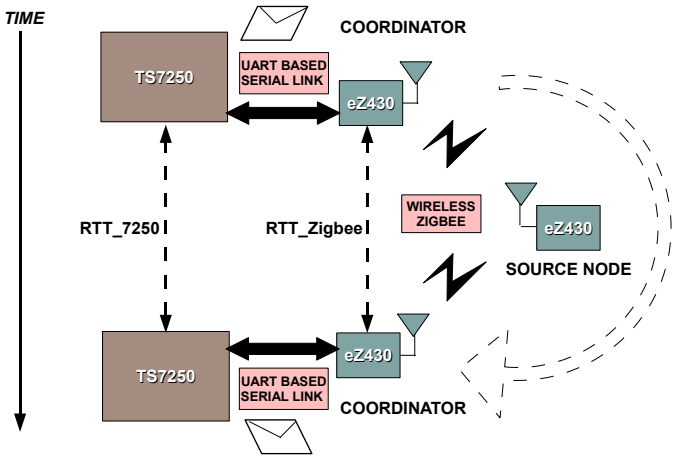
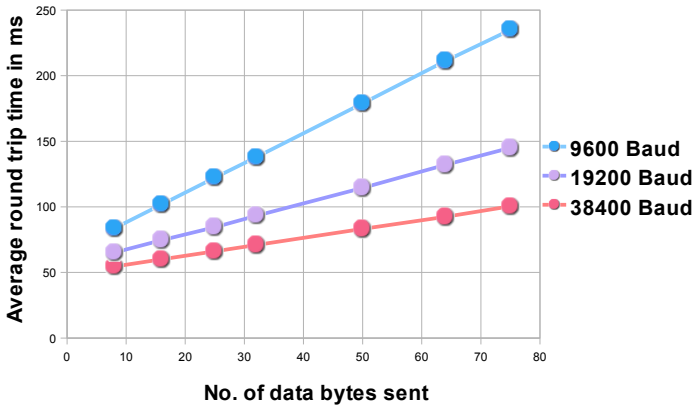


Figure 4.1: Round trip delay in the critical path of the RSC platform

Several experiments were done to find the round trip time (RTT) in the critical path, shown in figure 4.1. The delay was measured for three different UART baudrates, 9.6 KBaud, 19.2 KBaud and 38.4 KBaud. Different packet sizes from 8 bytes to 75 bytes were tried to observe the changes in delay. The application code on the eZ430 restricts the size of the buffers for the UART and Zigbee communication between 128 and 256 Bytes. This is done because the MSP430 has a RAM of approximately *1KB* and higher buffer sizes would consume excessive space, leaving little for other data. In addition, the eZ430

nodes are sensor nodes and are meant for communicating light-weight packets. Hence it was impractical to measure delays for the path with packet sizes more than 75 Bytes.
The resultant round trip times are plotted for each of the baudrates, against various packet sizes. The graph is shown in figure 4.2. 4.2(a) shows the overall round trip time of the path, shown in figure 4.1, from the TS7250 to the source node (via the coordinator node), and back. This is labeled as *RTT_7250*. As expected, the delay increases, with increase in the packet size and reduces, with the increase in baudrate. The increase in delay is linear since the packet sizes are increased linearly. However, the decrease in the slope of the delay is not linear, since the increase in the baudrates, is not linear. It is rather in powers of 2. Measurements were also taken to find out the round trip delay incurred, when the packet leaves the coordinator, reaches the source node, and comes back to the coordinator. This time is labeled as *RTT_eZ430* and is shown in figure 4.2(b). There could be several factors responsible for the delay, including the limitations of interface speeds and inherent wait and sleep times within the code running on the MSP430. The application code on the TS7250, is being run by an OS and there might be time delays caused, due to context switching and resource sharing. More time needs to be devoted to investigate the cause for the delays and this constitutes as future work that can be carried out in this project.

(a) RTT_7250 for various packet sizes and UART baudrates



(b) RTT_eZ430 for various packet sizes

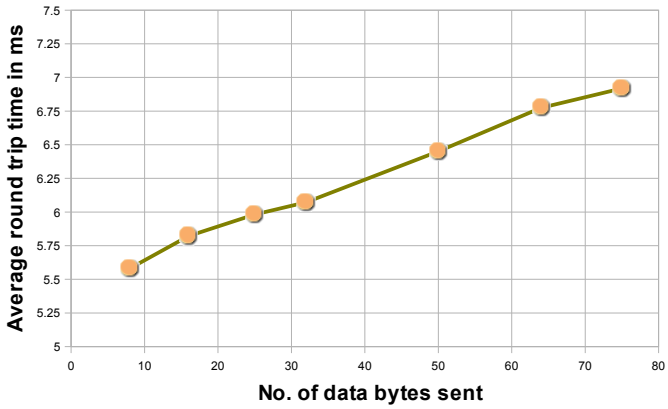


Figure 4.2: RTT measurements for the RSC platform

Chapter 5

The Rivalrous Hardware Scheduling (RHS) model as applicable to FREEDM Systems

5.1 Rivalrous hardware components

Many hardware components interfere with simultaneous functioning of one another, wherein, one generates electrical/magnetic noise that disturbs the functioning of another, sensitive to such noise. Such hardware resources can be termed as *Rivalrous Hardware*. A Switch Mode Power Supply (SMPS) [17] and Analog to Digital Converter (ADC) make a good example of rivalrous hardware because running an SMPS would create electrical disturbances that come in the way of an ADC, while it tries to take an accurate voltage reading. But in some real-time embedded systems, several of these rivalrous hardware need to be used simultaneously, within close proximity of one another. This forces designers to include expensive noise filtering components in the circuit in order to minimize interference among them. Not only do such filters consume more power, they also consume more area and hence add to the cost of the final design.

5.2 RHS model and its benefits

In real-time systems, the application consists of several tasks and these tasks are run based on some scheduling algorithm. On a uniprocessor system, they run in a mutually exclusive fashion. The usage of many hardware components, like ADC, is closely related to the software tasks running on the system. However, there are also components like the SMPS, which run independently, and not on the processor. By treating the run times of such independent resources as new tasks and by interleaving them with the existing 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). Figure 5.1 depicts the RHS model and the concept

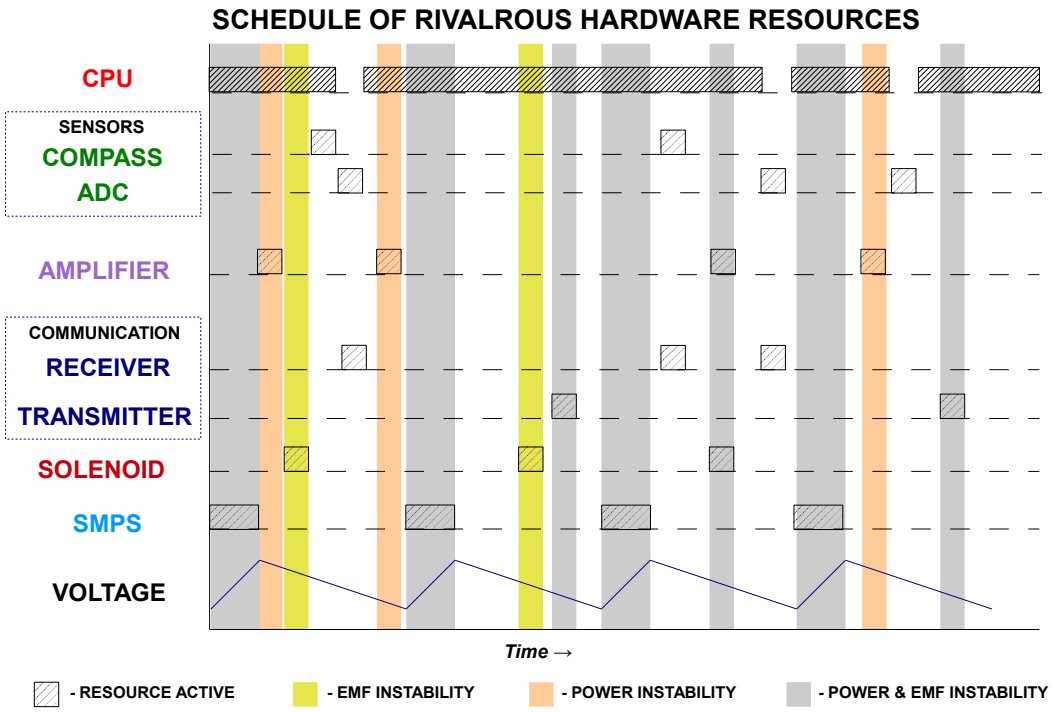


Figure 5.1: Timing diagram depicting usage and effects of running various hardware resources

of resource usage with respect to time. The figure describes the effects of running some well

known hardware components, as seen by other components. For example, running an RF transmitter would create EMF and power fluctuations and hence noise sensitive sensors like ADC or a compass cannot be run at this time. However, the CPU is not affected by such a disturbance and continues to run even while the RF transmitter is active.
The figure also shows operation of an SMPS and its impact on the supply voltage. Powering up an embedded board through an SMPS helps save power since it works at very low input battery voltages (say 1V) and steps up its output to normal operating voltages (more than 2V) of most embedded boards. An SMPS is useful while applying power optimization techniques like voltage scaling. But at the same time, using SMPS introduces both power and EMF instability in the system. With the adoption of RHS, an SMPS can be included into a system without incurring additional cost for filters.

5.3 Applying the RHS model on the RSC platform

5.3.1 Selecting the appropriate RSC hardware for RHS

The eZ430 nodes that form the lower tier of the RSC system, discussed in earlier chapters, form a good starting base to implement the RHS model because of several reasons:

1. The state machine based application code runs directly from Flash memory, instead of being handled by an OS (like in the TS7250 boards), and such an environment offers ease and flexibility to implement a scheduler that is tailor-made for RHS.

2. The eZ430 boards contain hardware components that are rivalrous in nature.

3. The nodes support voltage scaling and provide easy access in software, to do frequency scaling too.

5.3.2 SMPS and its benefits

A switch mode power supply (SMPS) uses a switching element to supply power to the load. Since a switching element is used, the efficiency of such a power supply is much higher, compared to a linear regulator, which incurs wastage of power through heat dissipation. [18] lists several types of SMPS, and provides more details and waveforms about each. For this discussion, the following three types are relevant:

1. Boost converter - Output voltage greater than input voltage.

2. Buck converter - Output voltage lesser than input voltage.

3. Split Pi or Buck-Boost converter - Output can be lesser or greater than input voltage.

A boost converter is capable of working with very low input voltages (1 V) and produce higher output voltages. It extracts maximum energy out of the batteries, thus extending their lifetime. Along with conversion, the SMPS in boost mode also regulates the voltage. So if need be, even an A.C. input could be used instead of D.C.

5.3.3 Motivation for adding an SMPS on the RSC platform

It is well known that the voltage of a battery continues to drop, with continuous usage, from its initial rated value (say 1.55 V), and reaches a point where its no longer useful, not because it does not contain charge, but for the fact that it does not produce the required operating voltage. In order to establish the fact that, using an SMPS to power up the MSP430 nodes, increases the life of batteries, as against running the nodes directly off batteries, or via a linear regulator, a couple of simulations were done. The circuits used for these hypothetical [1] simulations are shown in figure 5.2.

The figure depicts three different cases where in the MSP430 nodes are powered up

1. directly from batteries - 5.2 (a)

2. through a LT 3080 [19] linear regulator - 5.2 (b)

3. through a standard SMPS IC such as LT 1308 [20] - 5.2 (c)

The two main metrics that dictate the efficiency of the system are - the rate of decrease of battery voltage and the power consumption of the MSP430. These two factors were plotted against time (in hours), starting from the point where the batteries are fully charged, to the point where the battery voltage dips so much, that it is no longer useful. Figure 5.3 shows the graphs for the afore mentioned metrics, for all the three power supply circuits, using the data sheet of a standard AA battery, like [21], as a reference.

Figure 5.3 (a) reveals that the dip in the battery voltage is faster, if the MSP430 is

[1] The simulations were not physically conducted. The behavioral results are derived from ratings provided in datasheets and manuals for the MSP430, for standard AA batteries.

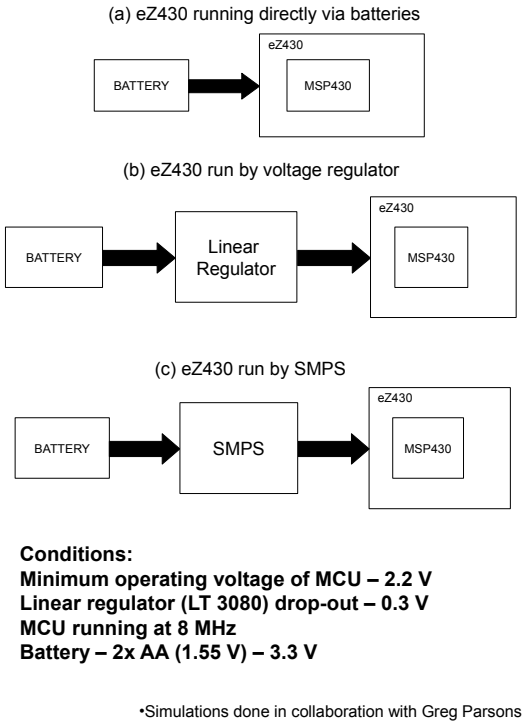


Figure 5.2: Simulation set up to compare regulated and unregulated power supplies

powered up directly and it could run for around 900 hours, until the voltage dips to 1.1 V. Even though the linear regulator and the SMPS follow a similar voltage dip, the SMPS outperforms the linear regulator because it continues to run for more than 1100 hours, even at battery voltages as low as 1 V, while the linear regulator stops much earlier, at around 760 hours, when the battery voltage reaches around 1.25 V. A linear regulator requires the input voltage to be above a threshold in order to deliver the desired output voltage and hence cannot function at voltages below 1.25 V. It is not power efficient either, since it dissipates excess energy as heat.

Figure 5.3 (b) shows the power consumption of the MSP, when plotted against the dipping battery voltage, over time, for all three cases mentioned earlier. From the plots, it can be observed that the MSP consumes maximum power when used with an unregulated power source and runs as long as 900 hours. The linear regulator reduces power loss to an extent,

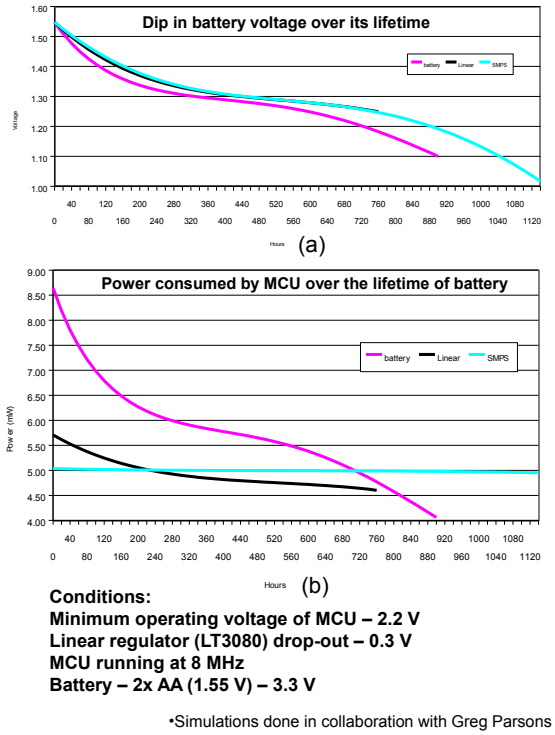


Figure 5.3: Drop in battery voltage and MCU power consumption, over time, for regulated and unregulated power supply

but fails to work after 780 hours. This is attributed to the fact that even though the linear regulator reduces power consumed by the MSP, the regulator itself burns substantial power to do so. This also means that, running the MSP directly from a battery leads to longer life than running on a linear regulator, and the same can be observed in both the graphs. However, it is clear that, the SMPS not only minimizes the power consumed by the MSP, but also increases the battery life to more than 1150 hours, which is much longer than the other two cases. The SMPS does so by consuming only what is required from the batteries and increases the overall power efficiency of the system. Such a benefit of reduction in power consumption, at a low additional cost, is a strong motivation to use the SMPS to power up the eZ430 nodes of the first generation RSC platform.

5.3.4 Using the RHS model to overcome the effects of adding an SMPS

Adding the SMPS induces EMI into the system, and this affects the normal functioning of devices such as ADC, Compass, etc, as discussed previously. However, this drawback is overcome by applying the concept of RHS, wherein the the run-times of the SMPS and other noise sensitive components are controlled using a scheduler or an OS. Thus the RSC platform can be made more power efficient at minimal cost. The implementation of the SMPS and the scheduling model, is described in the next few chapters.

Chapter 6

Adding a processor controlled SMPS to the first generation RSC platform

6.1 Principle of SMPS working

As shown in chapter 5, addition of SMPS greatly improves the life of the batteries and improves the power efficiency of the system. For the eZ430 nodes, a boost converter is most appropriate, for reasons mentioned in the previous chapter. The basic circuit needed to create the boost SMPS is shown in figure 6.1. The principle of working of a boost SMPS circuit can also be understood with the help of the same. *V-In* is the input battery voltage that drives the circuit. The inductor *L1* forms the reactive element, while the transistor *Q2* forms the switching element in the circuit. The diode *D4* allows charging of the capacitor *C1* and prevents discharging of the same.

When Q2 is turned on, the inductor is shorted to ground and currents builds up exponentially, as the inductor starts charging up. When Q2 is turned off, the current through the inductor ensures that the voltage across Q2 is higher than the capacitor voltage. Hence, diode D4 is forward biased and the charge stored in the inductor is pushed across to the capacitor C1 through D4. As the capacitor voltage V_{OUTPUT} rises, the voltage across Q2 reduces and at some point, charge can no longer flow across the diode. At this point, the diode turns reverse biased and ensures the capacitor does not discharge through it.

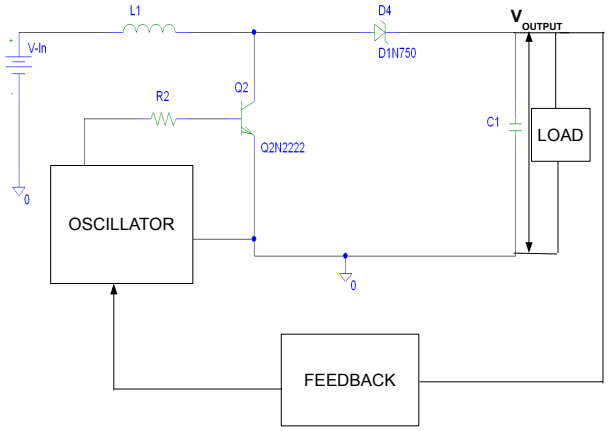


Figure 6.1: A basic SMPS circuit

By repeating this process, the voltage across the capacitor can be built up in steps. Hence Q2 is being driven by an *Oscillator*, whose frequency decides the rate of switching of Q2 and in turn, the rate at which C1 charges up. Thus, capacitor C1 forms a charge reservoir, as voltage builds up across it. This charge can be used to drive a *load*, in the present discussion, the eZ430 boards. Also seen in figure 6.1, is the *Feedback* module which controls the oscillator. This feedback turns on the oscillator when the output voltage V_{OUTPUT} dips below a certain lower threshold. This switches Q2 and the capacitor starts charging up. When V_{OUTPUT} reaches an upper threshold, the feedback circuit turns off the SMPS. This maintains the voltage across the capacitor, V_{OUTPUT}, within an acceptable range. This is useful when driving loads such as micro controllers on the eZ430. Such processors can operate only within a fixed range of supply voltages and would shutdown or burnout when their supply voltage goes too low or too high.

6.2 Oscillator for the SMPS

6.2.1 Design options for the oscillator circuit

There were several options to build an oscillator. A readily available IC like an *LT1303* [22], containing both the oscillator and an inbuilt feedback circuit could be used.

However, the cost of adding such an IC is pretty high and provides minimal flexibility in terms of final output voltage and oscillator frequency. Besides, that would make the circuit bulky and hence would consume more power and area. An Operational Amplifier (OpAmp) could be used in comparator mode, as an oscillator, as shown in [23]. But for low battery voltages, an OpAmp may not oscillate soon enough, or worse, it may not work at all and this breaks the integrity of the circuit. Adding an OpAmp is more expensive and power consuming too.

Considering the pros and cons of the above cases, the option of using an a-stable multi-vibrator [24] was found to be suitable and desirable. An a-stable vibrator converts D.C to A.C and can be easily designed with 2 transistors, 2 capacitors and 4 resistors. This circuit works with very low battery voltages (as low as 1 V) and hence stretches life of the batteries to the maximum extent, as discussed earlier. This suits the RHS model described in chapter 5. Additionally, the circuit is very flexible, where in, the duty cycle and the oscillating frequency of the output voltage, along with the output current delivered to the load, can be varied in finer steps, without major changes to the circuit. These variations in the frequency of the oscillator output changes the switching rate of transistor Q2, shown in figure 6.1 and the variations in the duty cycle of the output, varies the on time of Q2. As the on time of Q2 increases, the current flowing through the inductor also increases and vice-versa. This varies the rate of charging of capacitor C1. The inductor current also varies with the change in the base current of Q2, delivered by the output of the oscillator. This also influences the rate of charging of C1 and in turn, the rate of rising of V_{OUTPUT}. This proves the flexibility of the oscillator designed using an a-stable multi-vibrator. Also, the cost, the area occupied and the power consumed by such a circuit is relatively less because the circuit is built with only simple components.

6.2.2 Oscillator implementation

The oscillator circuit assembled for the SMPS is shown in figure 6.2(a). *VCC* is the power supply for the vibrator circuit and is also used as the enable signal. This is used by the feedback circuit to control the oscillator. In the circuit shown in figure 6.2, capacitors *C2* and *C3* charge up via *R7* and *R6* and discharge via transistors *Q3* and *Q4* respectively. The capacitors charge and discharge continuously, as long as VCC is high, but do so alternately. This means that when C2 is charging, C3 discharges and vice-versa. Since

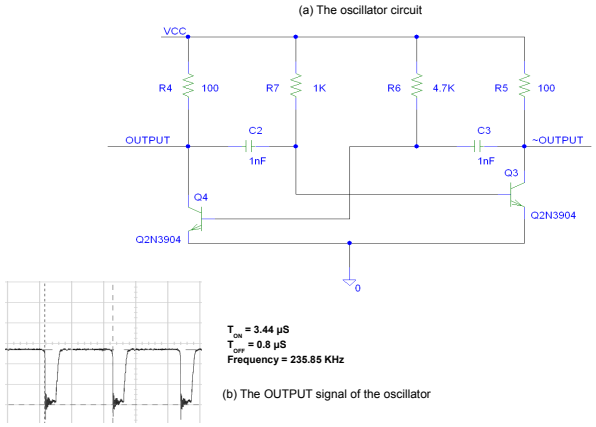


Figure 6.2: Oscillator for the SMPS circuit

C2 and C3 are connected to the base of transistors Q3 and Q4, respectively, this makes the transistors switch in alternate and opposite manner, relative to each other. This produces a continuous oscillating square waveform at the collectors of both Q3 and Q4. But the output voltage across them are complement of each other.

The value of R7, R6, C2 and C3 control the rate of charging and discharging of C2 and C3 and consequently control the duty cycle of the output waveform. The values of R4 and R5 control the output load current. If the value of R4 and R5 is equal to 100 Ω, as shown in figure 6.2, a load current of upto 20 mA can be supplied by the oscillator, at a VCC of 2V. For the values of R7, R6, C2 and C3 shown in the figure, the output produced has T_{ON} of 3.44 μs and T_{OFF} of 0.8 μs, at a frequency of 235.85 KHz. The output waveform can be observed in 6.2(b). Since the load current requirements for the oscillator circuit is relatively small ($<$ 100 mA), Q2N3904 [25] transistors are sufficient for Q3 and Q4, where as, for the switching transistor Q2 in 6.1, Q2N2222 [26] transistors are used, to handle inductor currents of upto 800 mA.

6.3 Processor controlled feedback for the SMPS

6.3.1 The feedback loop

An important aspect for the SMPS circuit is the feedback loop. With reference to figure 6.1, when the oscillator is on, and switching the transistor Q2, the output voltage V_{OUTPUT}, across capacitor C1, rises in steps. If this rise in voltage is not checked by turning off the oscillator at the right moment, the voltage will continue rising and exceed the maximum operating limits of the load, thus damaging the load circuit. On the contrary, if the SMPS is not switched on at the right moment, the load will continue to drain the charge in the capacitor, there by making V_{OUTPUT} dip below the minimum operating voltage of the load, eventually switching the load off. The feedback circuit tracks the rise and fall of the voltage across capacitor C1 and restricts variation of V_{OUTPUT} within a certain acceptable range, by turning the oscillator off or on accordingly. This range is often decided by the operating voltage of the load that is being driven by the SMPS.

The load for the SMPS circuit, as applicable to RHS, is the eZ430 nodes, and the SMPS design needs to meet specifications of the components present on the node. The eZ430 nodes contain the MSP430 and the CC2480 ICs and the combined maximum operating voltage supported by either ICs is 3.9 V. The combined minimum voltage needed to run the ICs is around 1.9 V, as evident from the datasheets [14]. So the feedback circuit had to be designed such that the oscillator would be turned on when V_{OUTPUT} dipped closer to around 2 V and turned off when V_{OUTPUT} rose up to around 3.8 V. The MSP430 supports frequency scaling and the main clock can be set to either 1MHz, 8 MHz, 12 MHz or 16 MHz. In order to support full range frequency scaling on the MSP430, V_{OUTPUT} needs to be raised to at least 3.3 V, before it is switched off. This is the minumum voltage at which MSP can can run at its maximum frequency, 16 MHz. But the maximum threshold before the SMPS switches off can be brought down if such a range of frequency scaling need not be supported. This helps conserve energy since the power used is proportional to V_2. Methods to vary maximum V_{OUTPUT} are discussed in the next section.

There are several ways to implement a feedback circuit. [23] shows a few examples. The rise and fall of V_{OUTPUT} can be compared against a fixed reference voltage, using a comparator, and the oscillator can be turned on or off based on the output of the comparator. The same principle is made use of, while implementing the feedback circuit for RHS. Additionally,

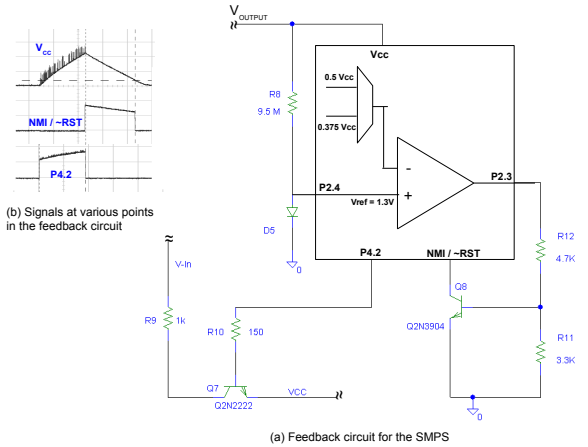


Figure 6.3: Feedback for the SMPS circuit

the feedback circuit is controlled and configured by the application software running on the MSP430. This medley of hardware implementation with software control makes the feedback system highly flexible. Also, since the load itself contains the major chunk of feedback circuit, the cost of additional feedback elements is eliminated.

6.3.2 Output voltage - sensing and comparison

The MSP430 contains an on-chip operational amplifier and is used for voltage comparison. The OpAmp can be used in several modes, one of which being a comparator, and the mode selection is software controlled. The implementation of feedback circuit is shown in figure 6.3(a). A constant reference voltage of around 1.3 V, obtained from a forward biased LED *D5*, is given as the input to the + terminal of the OpAmp. A big resistor *R8* (around 9.5 M Ω), is in series with D5 to prevent shorting V_{OUTPUT} to the forward bias voltage of LED D5. The OpAmp circuit on the MSP430 contains a resistor ladder (not shown in figure) internally, which forms a divider circuit, and can give several fractions of supply voltage Vcc to the input terminals of the OpAmp. A multiplexer selects what fraction of Vcc is to be fed into the - terminal of the OpAmp. This selection is software controlled and can be changed on-the-fly. In the present implementation, two different levels

- 0.5 x Vcc [1] and 0.375 x Vcc, are fed to the - terminal of the OpAmp in mutually exclusive manner. When the fraction of Vcc is below the diode reference of 1.3 V, the output of the OpAmp is high. As V_{OUTPUT} rises, Vcc rises, and at some point, as its fraction exceeds 1.3 V, the output of the OpAmp turns low. As V_{OUTPUT} falls, Vcc falls and the value at - terminal of OpAmp falls below that of its + terminal. This makes the output of the OpAmp go high again.

6.3.3 Driving the oscillator

The comparator forms the first half of the feedback circuit. The output of the OpAmp drives the *NMI/∼RST* pin of the MSP430. The signal on this pin generates a non maskable interrupt (NMI) which lets the software on the MSP realize when to write a *0* or *1* to the digital I/O (DIO) pin *P4.2* (bit 2 of port 4). The software along with the NMI and DIO pins form the second half of the feedback circuit. The NMI is edge triggered and the software can select the interrupt to occur on a rising or a falling edge, on-the-fly. Pin P4.2 of the MSP is connected to base of transistor *Q7* as shown in figure 6.3. The collector of Q7 is powered up by the battery voltage V-In shown in 6.1 and the emitter of Q7 is connected to the VCC of the oscillator circuit shown in 6.2. So when P4.2 is turned on by the software, Q7 is turned on. This powers up the oscillator and turns on the SMPS, and vice-versa. The signals at relevant pins are shown in figure 6.3(b). Hence Q7 acts as an enable switch for the oscillator. Transistor *Q8* is used for inversion of the OpAmp output and also to prevent false triggering of NMI. This completes the feedback loop.

6.4 Complete SMPS circuit and its working

The complete SMPS circuit assembled is shown in figure 6.4. It is a combination of figures 6.1, 6.2 and 6.3. Initially, V_{OUTPUT} is as low as 2 V. The software running on the MSP430 configures the NMI for a rising edge trigger and selects 0.375 x Vcc as the - pin for the OpAmp. This means that the voltage at the - terminal is around 0.75 V, for a Vcc of 2 V, which is less than the diode reference of 1.3 V, and hence the OpAmp output at pin P2.3 is high. This is inverted by Q8 and keeps the NMI pin low. When the software on the MSP430 decides to turn on the SMPS, it writes a 1 on the DIO pin P4.2. This

[1] read as 0.5 times Vcc, etc.

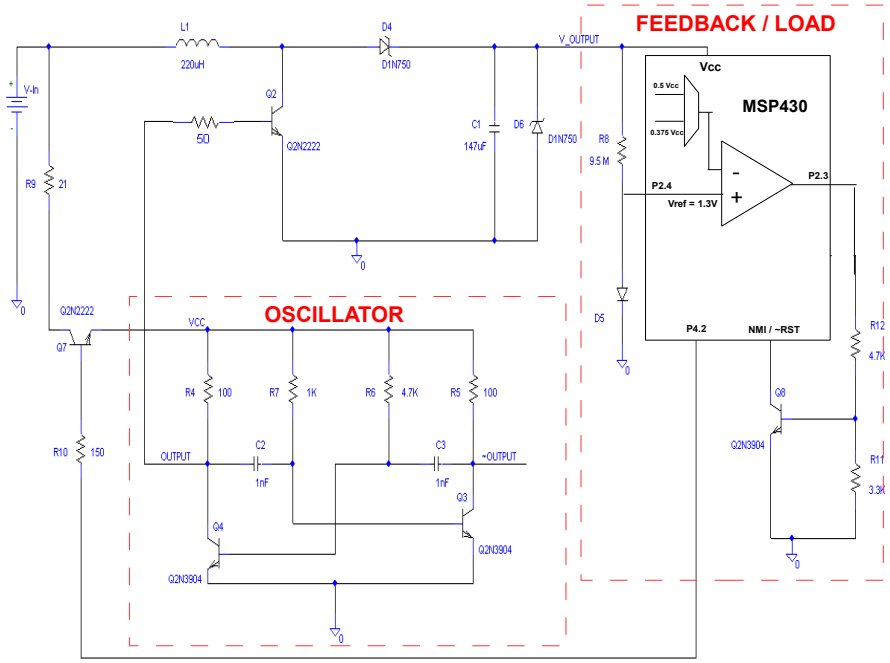


Figure 6.4: The complete SMPS circuit

turns on Q7 and subsequently the oscillator. The oscillating OUTPUT signal switches Q2 and this results in rapid change in the current through the inductor L1. As a result of the rapid change in the inductor current, capacitor C1 starts charging up and V_{OUTPUT} (Vcc of the MSP) starts rising. When V_{OUTPUT} reaches around 3.6 V, the voltage at the - pin of the OpAmp just crosses above the diode reference voltage of 1.3 V. This change drives the OpAmp output low and consequently changes the voltage at NMI pin from low to high. This rising edge at the NMI pin triggers an interrupt to the MSP and the NMI ISR writes a 0 onto pin P4.2 of the MSP. This switches off the oscillator and subsequently the SMPS. Once the SMPS is switched off, the software on the MSP, now configures the NMI to be triggered on a falling edge and selects 0.5 x Vcc as the input to the - pin of the OpAmp. Since the SMPS is off, V_{OUTPUT} continues to drop as the load draws power. As Vcc of the MSP dips below around 2.6 V, the voltage at the - pin of the OpAmp drops below Vref of 1.3 V and hence the output of the OpAmp now changes from low to high. This is inverted by Q8 and a falling edge occurs at the NMI pin. Again, an interrupt is generated in the

MSP and the ISR sets a flag, to indicate that the Vcc is dropping below 2.6 V, and the SMPS should be turned on. The ISR differentiates between when to turn on and off the oscillator based on the edge that triggered the NMI. However, the ISR does not turn on the SMPS right away. The software reads the flag set by the ISR, and turns on the SMPS, anytime before Vcc dips to 2 V. This mechanism gives the software some time to finish the task it is executing, before the SMPS can be turned on.

6.5 Control of SMPS with a scheduler

As per the RHS model, the run times of SMPS should be mutually exclusive, to that of the software tasks running on the system, which use noise sensitive components. The SMPS implemented for the RSC platform is capable of being controlled by a scheduler or an RTOS. In a way, the running of SMPS can be perceived as a real-time task in the system, whose run times are decided by the scheduler. This flexibility allows running tasks, that use hardware, rivalrous in nature to the SMPS, either before or after the SMPS task. When the SMPS runs, the scheduler can continue to run tasks, that do not conflict with the SMPS. By fine-tuning the SMPS circuit, the charging and discharging time of C1 can be made predictable. This time depends on the battery voltage V-In. As the charge in the battery reduces with usage, the charging time for C1 increases. A possible minimum can be chosen for the battery voltage, and the the charging time of C1 at this lowest voltage can be taken to be the Worst Case Execution Time (WCET) of the SMPS task. Similarly, the charging time of the C1 is lowest when V-In is high and the Best Case Execution Time (BCET) for the SMPS task can be found out at this maximum possible battery voltage.

Chapter 7

A scheduling approach to overcome SMPS interference

7.1 Options to reduce noise and interference

Any *source* [1] node, running the ZASA code, periodically senses temperature and voltage readings and sends them to the coordinator node. The coordinator node forwards the same to the TS7250 over the UART interface. The MSP430 uses ADC to derive the temperature and voltage readings. The accuracy of temperature and voltage readings is very important in the context of the RSC platform, since these readings are recorded constantly, and their variations are monitored. If the readings vary a lot, it may trigger false alarms within the monitoring system and this breaks the integrity of the energy management system. However, the ADC and SMPS form rivalrous resources, as already mentioned. In order to overcome rivalrous hardware bumping into each other's paths, several approaches were studied.

7.1.1 Hardware approach

Using a filter

There are various implementations of inductor and capacitor based filters [27], especially in rectifier circuits, that help minimize ripples in the supply voltage. An inductor,

[1] The various roles of the nodes in the network were briefly mentioned in chapter 2

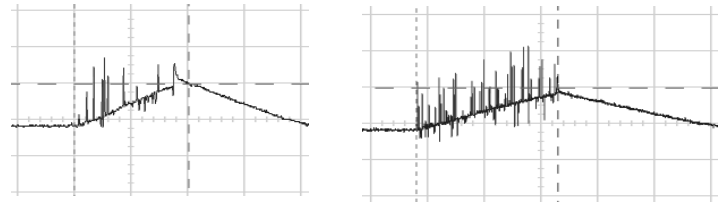
when connected in series, opposes change in its input voltage. Using a big capacitor helps increase the charging time, thus reducing ripples. If a load with a big resistance is used, the discharge time of the capacitor reduces as well, thus reducing the ripples to a larger extent. However using large L-C filters is not a good option for several reasons. These reasons are discussed for various filters in the following few sub sections.

Capacitor input filter

A capacitor input filter can be built by adding a capacitor in parallel to the output of the SMPS, as shown in [5]. The reactive component, X_C, of a capacitor is given by $X_C = \frac{1}{2\pi fC}$ [5]. This means that, as the frequency of distortions (f), in the voltage, increases, the capacitive reactance reduces. Since the SMPS circuit generates noise distortions in the KHz range, this filter shows poor regulation. Hence large capacitor values (C) are required to compensate for the loss in reactance. The area occupied by big value capacitors is high and this large physical size limits their practical use in several small scale circuits.
Figure 7.1 compares the noise factor and the rise times of SMPS output against various values of output capacitors. It can be observed that even though the output capacitor value is made larger in each steps, the distortions in the SMPS signal do not reduce. It can also be seen that increasing the capacitance increases the rise time of the output signal. Capacitor filters are more effective in reducing ripples of lower frequencies (a few Hz), like those present in A.C. power lines.

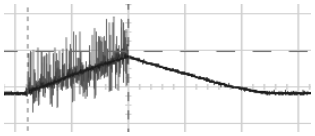
Choke input filter

Choke filters are another alternative, where an inductor is added in series, after the SMPS output followed by another capacitor in parallel. The reactive component, X_L, of an inductor is given by $X_L = 2\pi fL$ [5]. Although this results in better voltage regulation, due to the fact that X_L increases with increase in frequency (f), the magnitude of the output voltage delivered is very low, since the inductor is connected in series. Using an inductor is practical for low frequency A.C. signals. Considering the fact that the SMPS is switched at high frequencies to charge up the load capacitor, using an inductor in series may block most of this pulsating component and thus result in very large charge times for the capacitor. Besides, adding an inductor introduces magnetic field distortions in the

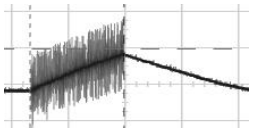


(a) C = 10 µF, Rise time = 400 µs

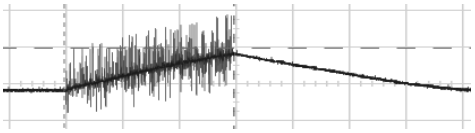
(b) C = 47 µF, Rise time = 1.25 ms



(c) C = 147 µF, Rise time = 3.56 ms



(d) C = 247 µF, Rise time = 6 ms



(e) C = 720 µF, Rise time = 16.6 ms

Figure 7.1: The rise time of SMPS output at various capacitor values

environment, adding to existing troubles. The cost factor also increases, with the addition of filter chokes.

High frequency filters

Articles [6] and [7] discuss about design of filters for SMPS in detail. Techniques to conquer differential (out of phase) and common mode (in phase) noises are also discussed. [7] also mentions about placing the sensitive components in the circuit, as far apart as possible, to reduce interference. This leads to larger board area and consequently an increase in cost. Considering the physical size of the eZ430 nodes and the magnitude of voltage it works at, using such expensive filter designs would be an overkill.

7.1.2 Software approach

For the eZ430 nodes, the operating voltage is low and the noise frequency is high. Using the filters discussed above is not practical since they occupy a large area and increase the cost of the circuit. At the same time, it is not possible to ignore the effects of EMI. One possible way to combat EMI is to use a software controlled mechanism. As mentioned in chapter 5, if the run times of such rivalrous hardware components can be controlled by the software, these devices can co-exist in close proximity. This requires use of real-time scheduling policies, in order to ensure that the hardware components run at the right time, and in mutually exclusive manner. To achieve this, a mathematical model had to be derived for the real-time tasks in the system.

7.2 The Make And Take (MAT) scheduling model

The addition of SMPS not only adds noise to the system, but also affects the run-times of tasks that are sensitive to EMI, since they have to be scheduled around the SMPS. Besides, the run-times of the SMPS itself is unpredictable since it depends on when the supply voltage dips below a certain threshold. Some tasks may use additional hardware components that draw more energy, resulting in the voltage dipping faster, while others may be pure software tasks that only run on the processor and draw only the amount of energy the processor needs.

The Make And Take (MAT) scheduling model not only considers the temporal parameters

of the tasks, but also the energy requirements of each task in the system. It bifurcates the tasks in any given system, into *Producers* and *Consumers* of energy. The MAT model helps compare the total charge produced against the total charge consumed in the system, in order to draw the upper and lower bounds for the run-times of the producers. Doing so, adds predictability to the run-times of components like SMPS. Once the temporal parameters of the producers are bound, they can be further interleaved with the temporal task model of the software tasks. The whole system can now be run with a suitable scheduling algorithm.

7.2.1 The temporal task model

A real-time system consists of various run time scenarios or use cases, called *Tasks* [28], whose properties depend on the application they are derived for. If a system contains N tasks, then all N tasks belong to a set T, such that, T = $\{t_1, t_2, .., t_N\}$, with priorities assigned to each task in non-increasing order. These tasks are composed of smaller execution units called *Jobs*. A task consists of one or many jobs, from a universal set of jobs J, which is the union of all jobs in the system. J = $\{j_{1,1}, j_{1,2}, ..j_{p,q}, j_{N,r}\}$ where, $j_{p,q}$ represents the q^{th} job of task p, such that, $1 \leq p \leq N$. $j_{N,r}$ is the r^{th} and final job of the last task t_N, in set T. Let K denote the cardinality [2] of J, while N denotes the cardinality of set T. It is assumed that any task t_i, for each $1 \leq i \leq N$, with a known period P_i would have a fixed maximum computation time C_i and a fixed relative deadline D_i, where $D_i \leq P_i$. The maximum computation time, also called the *Worst Case Execution Time* for any task t_i, is represented as $WCET_i$. All these assumptions still hold good for the MAT model.

Further, since the SMPS is now part of the scheduling model, it is also treated as another task in the system. The execution time for the SMPS is decided based on the time it takes to charge its output voltage to the desired level. However, this execution time depends on several factors, like the output capacitance, input voltage from the battery and the frequency of the oscillator. By specifying upper and lower bounds to these parameters, the execution time for the SMPS task can also be bound. Thus, the WCET for the SMPS task occurs when its output voltage is charged up from the minimum possible value to the maximum possible value, where the capacitance has the maximum possible value, the input battery voltage is at its lowest and the oscillator is running at the least possible frequency. This is represented as $WCET_{SMPS}$.

[2] The total number of elements in the set

7.2.2 Energy model of the system

For the sake of this discussion it can be assumed that the SMPS is the only producer task in the system, while all other software tasks are consumers. The introduction of SMPS into the system causes a swing in the output voltage and at the same time a lot of noise. This forbids the direct insertion of SMPS into the software task model, as discussed before. It is well known that, the voltage V, across the output capacitance C [3] of an SMPS, can also be treated as the equivalent amount of charge Q, contained in the capacitor [29]. This relation is given by

$$Q = C \times V \tag{7.1}$$

Based on this principle, all tasks in the system are divided into two categories - those that *produce* charge, like the SMPS, and those that *consume* charge, i.e. all other software tasks. Along with the WCET, the maximum charge consumed or produced by each task is also found out. This is represented as Q_MAX. Since the SMPS adds charge to the capacitor, the maximum amount of charge added by the SMPS is represented as Q_MAX_{SMPS}. Similarly, any software task t_i, draws charge from the capacitor while it executes and the maximum amount of charge drawn by any instance of t_i is represented as Q_MAX_i. The instance that draws Q_MAX_i need not directly correspond to the instance of the task which runs for the longest time, i.e. $WCET_i$. It depends primarily on the energy requirement of the hardware that each instance uses. In order to find Q_MAX_i, the supply voltage before and after executing the task is measured. Using equation 7.1, the corresponding charge present on the supply capacitor before and after the task execution is found out, say Q_START_i and Q_END_i respectively. The maximum difference between Q_START_i and Q_END_i gives Q_MAX_i [4]. In other words, $Q\_MAX_i = MAX(Q\_START_i - Q\_END_i)$. For the task set to be schedulable,

$$Q\_MAX_{SMPS} \geq Q\_MAX_i, \forall i \in N \tag{7.2}$$

In other words, the SMPS should be capable of meeting the maximum charge consumption of any software task present in the system. Equation 7.2 has to be always satisfied before

[3] Unless otherwise mentioned, from now on the voltage across C is called V_{OUTPUT} and since V_{OUTPUT} forms the supply voltage of the load, it is also called V_{SUPPLY}.

[4] While finding out Q_MAX_i, it is important to ensure that the SMPS does not run while the task is executing, since this affects the charge concentration on the capacitor.

the task set can be scheduled in a feasible manner. Failure to satisfy equation 7.2 would result in a situation where a particular instance of a task draws more charge than the SMPS could possibly provide, thus bringing the supply voltage below operating limits. This breaks the integrity of the system.
This energy model helps decide the upper and lower bounds for the SMPS, based on the largest Q_MAX in the system. Thus $WCET_{SMPS}$ for any given system depends on the largest value of Q_MAX_i, $\forall i \in N$.
For the SMPS, the following assumptions are made -

1. The supply voltage will never increase above a maximum value V_{MAX} and this translates to the maximum amount of charge the capacitor can contain.

2. The supply voltage will never drop below a minimum value V_{MIN}, which corresponds to the minimum operating voltage of the underlying hardware.

3. A threshold voltage $V_{THRESHOLD}$ exists such that $V_{MAX} > V_{THRESHOLD} \geq V_{MIN}$.

4. When the SMPS is not running and the supply voltage drops below $V_{THRESHOLD}$, an interrupt is generated to the MCU so that it can turn on the SMPS anytime before the voltage drops below V_{MIN}.

5. The difference between the charge on the capacitor at $V_{THRESHOLD}$ and V_{MIN} represents a charge buffer, Q_{BUFFER} in the system, i.e. $Q_{BUFFER} = Q_{THRESHOLD} - Q_{MIN}$.

6. Values for V_{MAX}, V_{MIN} and $V_{THRESHOLD}$ are selected based on the specifications of the underlying hardware.

Interleaving the temporal and energy models

Once the WCET and Q_MAX of the producers and consumers are found out, they can be used for scheduling. All software tasks are scheduled based on their temporal parameters. Even though the SMPS is considered a task, it is not explicitly included in the temporal schedule. However $WCET_{SMPS}$ is added to the WCET of all tasks which are rivalrous to the SMPS. Thus for any task t_j^*, which is sensitive to the noise produced by

the SMPS, $WCET_j$ changes to $WCET_j^*$, where $WCET_j^*$ is given by

$$WCET_j^* = WCET_{SMPS} + WCET_j \tag{7.3}$$

Equation 7.3 is only applicable for tasks that are not compatible with the SMPS. For tasks that can run in parallel with the SMPS, the WCET remains unchanged.
Thus the utilization of the task set also changes accordingly. If U is the original utilization of the task set before applying the changes according to the MAT model, U is given by

$$U = \sum(WCET_i/P_i), \forall i \in N \tag{7.4}$$

However, after applying the MAT model, the utilization changes to U^*, given by

$$U^* = U + \sum(WCET_{SMPS}/P_j), \forall j \in N \tag{7.5}$$

where t_j^* is any task that is sensitive to SMPS noise, as mentioned earlier.
The MAT model is independent of the actual scheduling algorithm used to run the tasks. After designing the system as per the MAT model, any suitable scheduling algorithm can be used to schedule the task set T, like Earliest Deadline First (EDF), Rate Monotonic (RM), etc.
However the scheduler needs to be changed slightly, so that it checks whether the SMPS is already running, every time it has to run a rivalrous task and delays the execution of the same until the SMPS has done charging fully. The SMPS task on the other hand is ready to run when V_{SUPPLY} dips below $V_{THRESHOLD}$. If no task is running, or is ready to run at that instant, the SMPS can be run straight away. But if a task is already running or is ready to run, the scheduler can be implemented such that, it can do one of the following things -

Start the SMPS immediately

If the scheduler gets an interrupt when V_{SUPPLY} dips below $V_{THRESHOLD}$ while a noise sensitive task is running, the task is paused and the SMPS is started immediately. If the task being executed is not sensitive, it is not paused and both are run in parallel. This is useful if Q_{BUFFER} in the system is not very big and there are very few tasks that are noise sensitive.

Postpone running the SMPS

The scheduler can track the charge used and the charge required by a task that is currently running. By storing the value of V_{SUPPLY} when the task started, and comparing it with $V_{THRESHOLD}$, the amount of charge already used by the task, Q_{USED} can be found out using equation 7.1. Thus, the charge required by the task, Q_{NEEDED} can be found out by $Q_{NEEDED} = Q\_MAX_i - Q_{USED}$. If $Q_{BUFFER} \geq Q_{NEEDED}$, the scheduler can postpone running the SMPS. Even though this makes the implementation of the scheduler slightly complex it results in good schedules. Having a big enough buffer is advantageous while running a task that is rivalrous with the SMPS. By postponing the running of SMPS, the rivalrous task can be run to completion rather than paused and resumed later.

Always run the SMPS when the task starts

If the underlying hardware is very small and the scheduler implementation is too simple to track charge usage at run-time, the SMPS can be run whenever any instance of any task starts to execute, whether V_{SUPPLY} is below $V_{THRESHOLD}$ or not. This ensures that the capacitor always has enough charge to sustain the task that is being run. However, if the task is noise sensitive, the SMPS has to run first and after it completes, the task is run.
The option that the scheduler takes at this point depends purely on the scheduler implementation, underlying hardware and the charge buffer Q_{BUFFER}.

7.2.3 A case study to understand the MAT model

To better understand the MAT model, consider a system comprising of three software tasks along with the SMPS task. So, T=$\{t_1, t_2^*, t_3\}$ and N=3. Let us assume that only t_2^*, is rivalrous with the SMPS and is annotated with a *. As per equation 7.3, WCET of the SMPS will be added to $WCET_2$ while scheduling t_2^*. The final values of WCET for each task is shown in table 7.1. Using the assumptions made by the MAT model, and the WCET shown in table 7.1, the tasks of T can be scheduled as per any scheduling algorithm. Figure 7.2 shows a snapshot of the tasks when they are run as per the MAT model. The top half of the figure shows the variations in the supply voltage V_{SUPPLY}, when the SMPS and the software tasks are run in tandem. The bottom half of the figure shows the run-times

Table 7.1: WCET for the example task set

Task	WCET
t_1	$WCET_1$
t_2^*	$WCET_2 + WCET_{SMPS}$
t_3	$WCET_3$

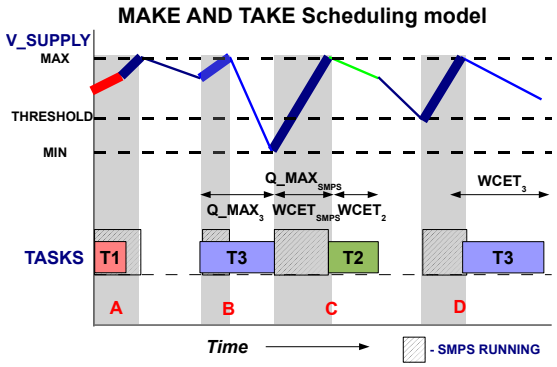


Figure 7.2: A sample schedule of tasks as per the MAT model

of the tasks and the SMPS. The shaded area overlapping both the top and bottom half represents the times when the SMPS is active. It can be observed that V_{SUPPLY} increases when the SMPS is running and decreases when any software task is running. When both the SMPS and the software task run at the same time, it can be seen that the slope for V_{SUPPLY} is still positive, but it is not as steep when the SMPS alone is running. This translates to the fact that the SMPS is adding charge at a faster rate, than the rate at which any software task is consuming. This satisfies equation 7.2, discussed previously. Figure 7.2 also shows four interesting scenarios that can occur at any point while running the tasks as per the MAT model. These are labeled *A, B, C* and *D* at the bottom of the figure.

Scenario A

Task t_1 and the SMPS both start running at the same time. Thus t_1 consumes charge and the SMPS adds charge at the same time. The slope of V_{SUPPLY} (portion shown in red) is small, but still positive. Although t_1 finishes, the SMPS continues to run and the slope of V_{SUPPLY} (portion shown in dark blue) now increases. The SMPS stops when V_{SUPPLY} reaches V_{MAX}.

Scenario B

At some point later in time, t_3 is released. The SMPS is already running when t_3 is released. However, since t_3 is not noise sensitive, it is dispatched for execution straight away. The SMPS finishes charging and stops before t_3. This time t_3 continues to run and this particular instance of t_3 draws the maximum amount of charge possible (portion shown in light blue), thus dropping V_{SUPPLY} to V_{MIN}. This represents Q_MAX_3 with reference to the MAT model. However before t_3 finishes, an interrupt is generated when V_{SUPPLY} drops below $V_{THRESHOLD}$. As discussed before, the action taken by the scheduler at this point can vary based on the implementation.

Scenario C

The scheduler chose to run t_3 to completion. t_3 eats deep into the buffer and even though t_2^* is ready as soon as t_3 finishes, it cannot be run since the SMPS has to be run first. The SMPS runs by charging up V_{SUPPLY} from V_{MIN} to V_{MAX} (portion shown in dark blue). This represents the $WCET_{SMPS}$ for the SMPS. It also corresponds to the maximum amount of charge produced by the SMPS, Q_MAX_{SMPS} as shown in the figure. t_2^* is finally dispatched for execution when the SMPS is done charging. This particular instance of t_2^* runs for the maximum amount of time, which is $WCET_2$ (portion shown in green). However if the wait time is added, the total run-time for t_2^* is $WCET_2 + WCET\_SMPS$, as represented in table 7.1. Since the task set is already scheduled for this WCET for task t_2^*, neither t_2^* nor any of the other tasks miss their deadlines.

Scenario D

After t_2^* finishes, even though there are no tasks running, V_{SUPPLY} continues to drop gradually due to leakage currents. Once the voltage dips below $V_{THRESHOLD}$, the scheduler receives an interrupt for the same and since no task is running, the scheduler runs the SMPS (portion shown in dark blue). When SMPS is almost done, t_3 is ready and starts running immediately. This instance of t_3 runs the longest with $WCET_3$ as shown in the figure (portion shown in light blue). Thus, even though this instance runs the longest, it does not require maximum energy Q_MAX_3, as discussed in the earlier section.

7.3 Running the tasks

7.3.1 Use of an RTOS

Using a real-time operating system (RTOS) to control the run times of rivalrous hardware is an option worth considering. Given the fact that it would have to run on the MSP430 chip of the eZ430 nodes, the underlying architecture of the MSP430 nodes had to be considered for choosing the right OS. Some of the guidelines drawn were:

1. The OS should be small enough to work with 1KB RAM and 32KB Flash.
2. It should support non-preemptive or cooperative scheduling.
3. The OS source code should be MSP430-GNU tool chain compatible.
4. It should preferably be open source, with reasonable documentation for the source code.

A survey was done, among some of the readily available real-time operating systems, to find the suitable one, satisfying most of the above criteria. The results of the survey are listed in table 7.2. It is evident from the table that none of them met the relevant criteria and hence could not be ported on to the MSP430. However, since the ZASA sample code, mentioned in earlier chapters, also contain source files that manage hardware resources, by acting as a hardware abstraction layer, the concepts of RHS can still be applied on these nodes, by adding just a scheduler, instead of a bulky operating system.

Table 7.2: Compatibility chart of real-time operating systems with MSP430 nodes

OS	**Fits into memory**	**Non-preemptive scheduler**	**GNU tool compliance**	**Open source**	**Overall compatibility**
μCOS-II	√	X	√	√	X
Salvo OS	√	X	X	X	X
Contiki OS	X	X	√	√	X
Free RTOS	X	X	√	X	X
Emb OS	√	X	X	X	X

7.3.2 Selecting a scheduling algorithm

Rate-Monotonic (RM) and Deadline-Monotonic (DM) are well known static scheduling algorithms that use fixed priorities to decide which task to run first. The priorities are assigned to the tasks at compile time, based on the shortest period first, or earliest deadline first methodology. EDF is optimal only when preemption is allowed. Preemptive scheduling algorithms are efficient and capable of scheduling a high proportion of task sets. However, they rely on context switching, which puts a high demand, on the resources of the underlying hardware, in terms of memory and processing power. Non-preemptive and static schedulers on the other hand are less resource hungry. Considering the limited RAM (∼1KB) and low computing power of the MSP430, a non-preemptive scheduling algorithm would be most appropriate to run the task sets on the eZ430 nodes. This reduces context switching costs and overhead. Hence a fixed priority rate monotonic run-to-completion scheduler is most suitable to run tasks using the MAT model on the RSC platform.

Chapter 8

Experiments and results

8.1 Experiments conducted

8.1.1 Demonstrating the concept of RHS

Tasks for the RSC platform

The ZASA code base was restructured suitably into several real-time tasks. Five independent tasks were identified for this system. Thus 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 t_2^* and t_3^* were identified to be rivalrous to SMPS.
As per the MAT model, once the tasks were identified their worst case execution times were found out. A *Profile Table* was created to record the WCET for each task. Each row in the profile table represented the worst case execution time for a particular task. The value in the row was only updated if the run time of its latest instance, was greater than the run times of all its previous instances. In order to find out the run time of tasks, a timer on the MSP430 was configured, with known settings. Macros were added at the beginning and the ending of the code section, representing the tasks, to start and stop the timer, respectively. At the end of the code section, the timer count was read, to obtain the number of timer ticks spent, while executing that particular instance of the task. If this value was greater than the already existing entry for that task, the new value replaced the old one in the row corresponding to the task, inside the profile table.
Since these embedded nodes have limited memory and processing capabilities, a simple test

Table 8.1: The tasks set for ZASA

Task ID	**Task Name**	**Maximum charge (μC)**	**WCET (ms)**
t_1	Application timer overflow	131.298	3
t_2^*	Obtain ADC samples of temperature and voltage	47.780	1
t_3^*	Send ADC samples using Zigbee	668.375	22
t_4	Check for correspondence from CC2480 chip	481.399	13
t_5	Button pressed on eZ430 board	123.321	3
t_6	Run SMPS	1221	10

was conducted to ensure that the maximum charge requirements of any task did not exceed the maximum charge produced by the SMPS. The SMPS was run before every instance of any task and after the supply voltage reached V_{MAX} the instance of the task was dispatched. After the task finished execution the supply voltage was read using the ADC. A variable in the task table stored this voltage and was only updated if the new value was lesser than the existing value. After the profiling period, this variable was read for each task and verified that it was greater than V_{MIN}. The SMPS was configured appropriately so that $WCET_{SMPS}$ was within acceptable limits. The values for V_{SUPPLY} were : V_{MAX} = 3.5 V, $V_{THRESHOLD}$ = 2.5 V and V_{MIN} = 2 V. These values were selected based on the safe operating voltage limits listed in the MCU datasheet.

The final task set for the ZASA code is shown in table 8.1. The SMPS is also listed as one of the tasks. $WCET_{SMPS}$ is listed as 10 ms and this occurs when the SMPS charges the supply voltage from V_{MIN} to V_{MAX} at a battery voltage (*V-In* of figure 6.4) of 2.3 V, oscillator frequency of 450 KHz and an output capacitance of 814 μF. All software tasks consume charge and the SMPS produces charge. The maximum amount of charge produced or consumed by the tasks are also listed in μC based on the outcome of the energy profiling tests conducted. Since the maximum charge produced by the SMPS is greater than maxi-

mum charge consumed by any task, equation 7.2 is guaranteed to be satisfied.
Considering the limited RAM space (1KB) on the MSP430 nodes, a non-preemptive scheduler was used to run the tasks in order to avoid the demands of context switching. For the scheduling algorithm, the fixed priority Rate Monotonic (RM) algorithm was chosen 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.

8.1.2 Using RHS to remove the interference of SMPS

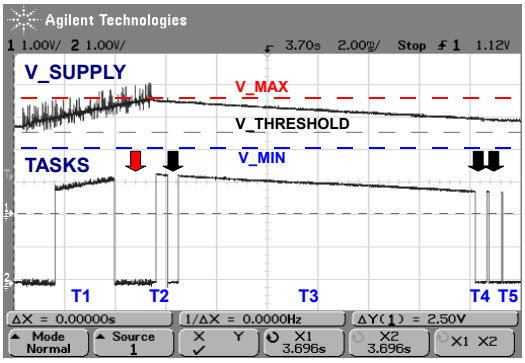
As previously discussed, the SMPS injects a lot of noise into the supply voltage. Since Task t_2^* used the ADC to read the temperature, experiments were conducted to study the impact on t_2^*, when it was run in parallel with the SMPS. The RHS concept of mutual exclusion was used to remove the interference between the SMPS and the ADC. The ADC samples before and after applying RHS were collected and plotted on a graph to observe the differences.

8.2 Results and Analysis

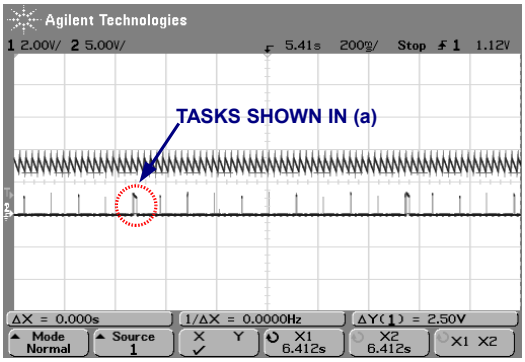
8.2.1 Waveforms for the SMPS and software tasks

The waveforms for the run-times of the SMPS task and all the software tasks were recorded from an oscilloscope to 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 8.1(a) shows a section of the schedule that clearly demonstrates the concept of RHS and also how the MAT model functions. In a way, 8.1(a) is a real-time replica of the MAT model depicted in figure 7.2. The top half of the figure shows the output voltage of the



(a) SMPS and task execution as per RHS and the MAT model



(b) Several instances of tasks spread out over a period of time

Figure 8.1: Waveforms to demonstrate RHS

SMPS, and all the three voltage levels - V_{MAX}, V_{MIN} and $V_{THRESHOLD}$ have been marked explicitly on the waveforms. The bottom half of 8.1(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, where as the red arrow indicates the instant where, t_2^* being incompatible with the SMPS, has to wait for the SMPS to finish charging up the voltage to V_{MAX} before it can start executing. 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

Table 8.2: Utilization for the ZASA task set

Task	WCET (ms)	$WCET^*$ (ms)	Period (ms)	U_i	U_i^*
t_1	3	-	100	0.03	-
t_2^*	1	11	100	0.01	0.11
t_3^*	22	32	100	0.22	0.32
t_4	13	-	200	0.065	-
t_5	3	-	200	0.015	-
SMPS	10	-	-	-	-

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.
Thus the execution of t_1 and t_2^* correspond to scenarios A and C explained earlier, with the use case example. Although in this case, the SMPS does not run for $WCET_{SMPS}$ before t_2^* begins. Figure 8.1(b) is a compressed form of 8.1(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.

8.2.2 Utilization of the task set before and after applying the MAT model

The temporal values for the ZASA task set used in the experiment is shown in table 8.2. The WCET for the SMPS task is also listed but is not considered while deriving the utilization, since the SMPS does not run on the MCU. The utilization of each task before and after applying the MAT model is also listed in 8.2. Since t_1, t_4 and t_5 are not rivalrous with the SMPS their WCET and utilization do not change. Using equation 7.4 the total utilization U, before applying the MAT model is U = 0.34. Using equation 7.5,

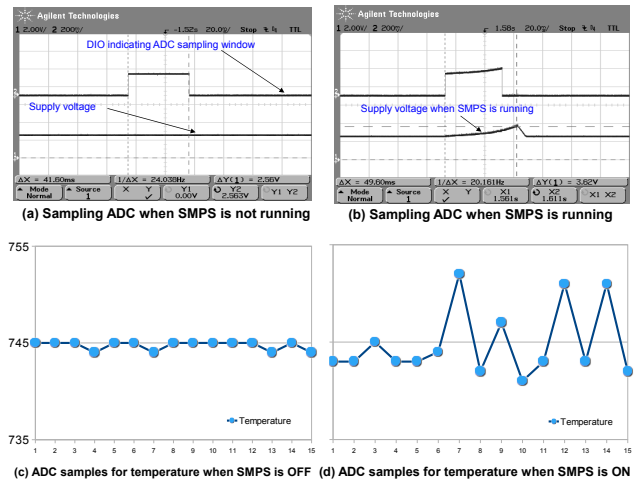


Figure 8.2: Impact of noise generated by SMPS on ADC

the total utilization U^*, after applying the MAT model becomes $U^* = 0.54$. Thus the total utilization of the system increases by 0.2. The increase in utilization of a rivalrous task is greater if $WCET_{SMPS}$ is much greater than the WCET of the rivalrous task, as in the case of t_2^* (90.9%). However it is not that large if $WCET_{SMPS}$ is lesser than WCET of the rivalrous task, as in the case of t_3^* (31.25 %). Thus if $WCET_{SMPS}$ can be made as small as possible, its negative impact on the schedulability and utilization of the software tasks can also be reduced.

8.2.3 Restoring normal functionality of ADC using RHS

In the ZASA 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 8.2 shows the waveforms for the experiment conducted in order to study the same. Figures 8.2 (a) and 8.2 (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. Fifteen consecutive samples from the ADC were recorded and plotted on a graph to study

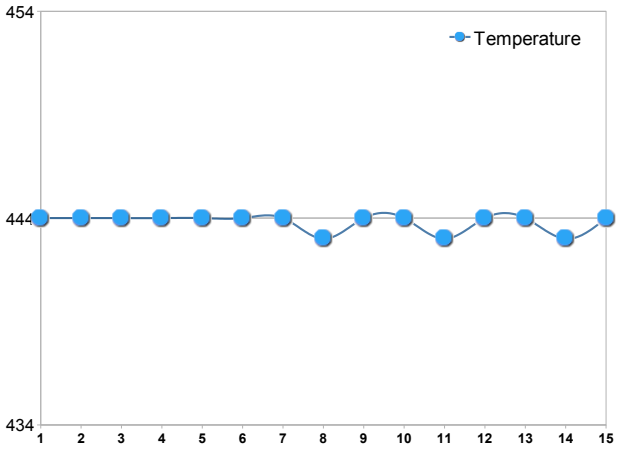


Figure 8.3: ADC samples for temperature with the SMPS running

variations within them. Figure 8.2 (c) shows the plot of these raw [1] 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 8.2 (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 8.3. 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.

[1] Here 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.

Chapter 9

Future work and conclusion

9.1 Future work

There is good scope to implement more functionalities onto the work done so far and optimize existing ones. These are presented in future sections.

9.1.1 Analysis on the MAT model

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. Based on these results and analysis, the SMPS can be optimized and the task sets that are most compatible with the MAT model can be found out. Doing so would give a better insight into the types of real-time systems for which RHS can be applied in the most efficient way. The schedulability of the task sets can be improved by using more optimal algorithms like EDF, instead of RM. One possible algorithm is Clairvoyant EDF (CEDF) [30] and the design for implementing the same has been presented in Appendix B. Also, the concept of RHS can be extended to several other combinations of rivalrous hardware components.

9.1.2 SMPS operation

The SMPS circuit implemented for RHS could be improved. At present, the circuit works for battery voltages as low as 2.3 V. The circuit could be optimized further to make the SMPS work with input battery voltages as low as 1 V. This would require the SMPS to start running at battery power up and raise the output above 2 V so that the eZ430 nodes can get started and take control of the SMPS. However, this new start up circuit has to be designed such that the output voltage does not rise very quickly. It may burn out the eZ430 nodes even before they start up and take control of the SMPS. Also, the output of the SMPS could be made to increase in steps, rather than one fixed maximum threshold, before it turns off. This could add flexibility and ensure that voltage is not raised higher than what is required. This would also save energy, since power burned by the eZ430 nodes is proportional to V^2. Such performance optimizations can be undertaken as part of future work.

9.1.3 Topology of networks in the RSC platform

The topology of network in both the top and bottom tiers of the RSC platform needs to be studied in more detail. Several well known topologies like star, mesh, etc. can be tried out for the nodes and the performance of the system can be recorded for each. In a star network, the center node has to do a lot of communication and if it is in the critical path, it would slow down the entire system. In a mesh network, even though nodes are spread out, keeping track of which nodes are still part of the network and which are not, could be tricky, especially if there are a large number of nodes. Experiments related to the afore mentioned scenarios and many more could be carried out to test the flexibility and improve the performance of the system

9.1.4 Time synchronization for eZ430 nodes

As mentioned in chapter 3, implementing time synchronization between the eZ430 nodes could be very useful. Data could be sent along with time stamps to improve the monitoring capability of the system. However, doing this could be challenging based on the granularity of the time. The MP430 chip contains several clock sources, which range from lower frequencies (KHz) to higher frequencies (MHz). If higher frequency clocks are used,

the granularity of time could be as low as 1 μ s, but when the MSP is put in low power mode, these clocks are switched off. This could affect the accuracy of time. The granularity of time required for the RSC platform and options to achieve such granularities need to be studied in greater depth.

9.2 Conclusion

A first generation communication platform, called RSC, was set up for smart energy management for FREEDM systems center. This platform supports a wide variety of interfaces and architecture. The nodes support various communication protocols and are capable of supporting more in future. This makes the platform reliable, robust and flexible. Having established the connections, experiments were conducted to study the performance to the communication system. Measurements were done with respect to round trip times, range, etc. to understand the capacity of the platform, in terms of packet sizes, data rates and distances, it supports.

The concept of Rivalrous Hardware Scheduling (RHS) was introduced, which involves controlling the run-times of rivalrous hardware through software, in order to avoid electromagnetic interference among them. 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, was also introduced. The concept of RHS was demonstrated and proved by building a processor controlled SMPS on a suitable platform and scheduling its run-times using the MAT model. Suitable waveforms and graphs were obtained to show the same. The utilization of the system before and after applying RHS were derived and it was found that even though the total utilization of the system increases after adding the SMPS, it can still be kept within acceptable limits by reducing the WCET of the SMPS, there by maintaining the schedulability of the task set. We also studied the impact of adding an SMPS on the ADC and showed how RHS can be successfully applied to remove the interference.

Bibliography

[1] *FREEDM Systems Center.*

[2] Load management. *Wikipedia*, 2010.

[3] Nikos Hatziargyriou. Microgrids large scale integration of micro-generation to low voltage grids. 2005.

[4] Bieshoy Awad Nick Jenkins and J.B. Ekanayake. Intelligent load control for frequency regulation in microgrids. 2008.

[5] Richard Lee Ozenbaugh. *EMI Filter Design.* Marcel Dekker, New York, NY, USA, 2001.

[6] D.H. Liu and J.G. Jiang. High frequency characteristic analysis of emi filter in switch mode power supply (smps). In *Power Electronics Specialists Conference, 2002. pesc 02. 2002 IEEE 33rd Annual*, volume 4, pages 2039–2043, 2002.

[7] I. Cadirci, B. Saka, and Y. Eristiren. Practical emi-filter-design procedure for high-power high-frequency smps according to mil-std 461. *Electric Power Applications, IEE Proceedings* -, 152(4):775–782, July 2005.

[8] Larry D. Bradley. Method and apparatus for feedback control of smps to ldo regulators, February 2000.

[9] M.A. Sorescu. Real time microprocessor control loop for switched mode power supply. In *Semiconductor Conference, 1997. CAS '97 Proceedings., 1997 International*, volume 2, pages 613–616 vol.2, Oct 1997.

[10] R. Pellizzoni, B.D. Bui, M. Caccamo, and Lui Sha. Coscheduling of cpu and i/o transactions in cots-based embedded systems. In *Real-Time Systems Symposium, 2008*, pages 221–231, 30 2008-Dec. 3 2008.

[11] Jianqing Zhang and Carl A. Gunter. Iec 61850 -communication networks and systems in substations. 2008.

[12] *Distributed Network Protocol*.

[13] Technologic Systems. *TS7250 - Hardware Manual*, 2008.

[14] Texas Instruments. *User's Guide*, 2008.

[15] *Zigbee Alliance*.

[16] *Simple Network Message Protocol*.

[17] Switch mode power supply. *Wikipedia*.

[18] Texas Instruments. Power supply technologies. 2008.

[19] Linear Technology. *LT3080 Datasheet*.

[20] Linear Technology. *LT1308 Datasheet*.

[21] Duracell. *AA battery datasheet*.

[22] Linear Technology. *LT1303 Datasheet*.

[23] Bryan Kris. Intelligent smps goes digital. 2009.

[24] Multi-vibrators. *Wikipedia*.

[25] Fairchild. *2N3904 Datasheet*.

[26] Fairchild. *2N2222 Datasheet*.

[27] Randall Aiken. Chokes explained. 2007.

[28] Jane S. Liu. *Real-Time Systems*. Prentice Hall Inc., Upper Saddle River, NJ, USA, 2000.

[29] James W. Nilsson and Susan A. Riedel. *Electric Circuits*. Prentice Hall Inc., Upper Saddle River, NJ, USA, 2000.

[30] C. Ekelin. Clairvoyant non-preemptive edf scheduling. In *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 7 pp.–32, 0-0 2006.

Appendices

APPENDIX A

Setting up the development environment for RSC

Appendix A, describes the methodology to set up the environment to implement the RSC platform and apply RHS principles on the same. The high end embedded board used for RSC is ARM based, whose specifications are listed at
$http://www.embeddedarm.com/products/board-detail.php?product=TS-7250$
Most of the documents and manuals for the board and the processor are available under the 'Resources' tab in the link above. For low level communication over zigbee, the eZ430-RF2480 kit from TI is being used. The specifications for the board can be found at
$http://focus.ti.com/docs/toolsw/folders/print/ez430-rf2480.html$
The 'Support software' section of the page in the above link provides the source code to be put on the MSP430 and an application s/w to be installed on the PC. The IDE provided by IAR, to develop and deploy code for the MSP430 has code size limitations and requires licenses to utilize all the features. Therefore it is recommended that the MSP430 GNU tool chain be used instead of the IAR. These steps can be followed:

1. Download a patch file from
   $http://jmkikori.homepage.bluewin.ch/devel/ez430-rf2480.html$
   which modifies the files in the ZAccel source code appropriately. If the patch file does not work, the files may have to be manually changed according to the patch. However, the patch will surely generate the required Makefile to get things started.

2. The MSP430-GNU tool chain needs to be set up on the host machine. The following wiki page shows how to set up the tool chain in windows if not already done:
   $http://wikis.lib.ncsu.edu/index.php/MSP430gdb$
   The page also shows commands needed to download and run the elf file using MSP430-GDB.

It is also important to note that there are various configurations done for the MSP with the 'DEFINE' flag in the Makefile, which have to be changed suitably. For ex: If the UART of MSP does not support baud rate of 115200, which is present by default in the Makefile the patch generates, it has to be changed to say 9600 before compiling the source code. Otherwise the UART of the MSP will not communicate properly.

APPENDIX B : Clairvoyant Non-preemptive EDF (CEDF)

Appendix B describes a design methodology developed to implement a non-preemptive algorithm called Clairvoyant EDF (CEDF). CEDF has a higher optimality compared to rate Monotonic (RM) algorithm. Due to time constraints, the following scheduling algorithm was not included as part of the project. The RM algorithm can be replaced by CEDF in the future, in order to run the ZASA tasks, thereby improving the performance of the system.

Introduction

The best way to implement non-preemptive EDF on the eZ430 nodes is to apply the methodology described in [30]. This paper proposes *Clairvoyant EDF (CEDF)* which is built on the non-preemptive EDF algorithm, but incorporates certain modifications to improve its schedulability. Under CEDF, the scheduler knows release times of tasks *a priori*. It will postpone execution of a ready task t_i, if a condition exists such that, running t_i at present time may cause deadline miss for another task t_j, arriving at a later time, but with an earlier deadline than t_i. The scheduler inserts idle times until t_j arrives. This postpone is only done when certain conditions are satisfied. By doing the postponement of certain tasks, CEDF ensures schedulability of some task sets that are not schedulable by EDF.
[30] also contains an experiment section that shows how the CEDF performs against both EDF [1] and the optimal EDF schedulers. The results suggest that CEDF schedules more task sets than EDF, for a system containing upto 50 tasks. Though CEDF does not outperform the optimal EDF, it still matches the performance of the optimal EDF, when the system has less than 20 tasks. For ZASA, the maximum number of tasks that currently exist is 6. Going by the results of the experiments, it is evident that CEDF could still be used for ZASA without any loss of performance. This would hold good even if more tasks are added later. Using CEDF not only ensures maximum schedulability of task sets, but also avoids costs of context switching and other overheads that come with preemptive scheduling algorithms, as discussed before. Hence CEDF seems like the best approach to implement real-time scheduling for ZASA tasks.

[1] From now on, the word 'EDF' would implicitly refer to non-preemptive EDF whereas 'optimal EDF' would refer to preemptive EDF, unless mentioned otherwise

Design and implementation

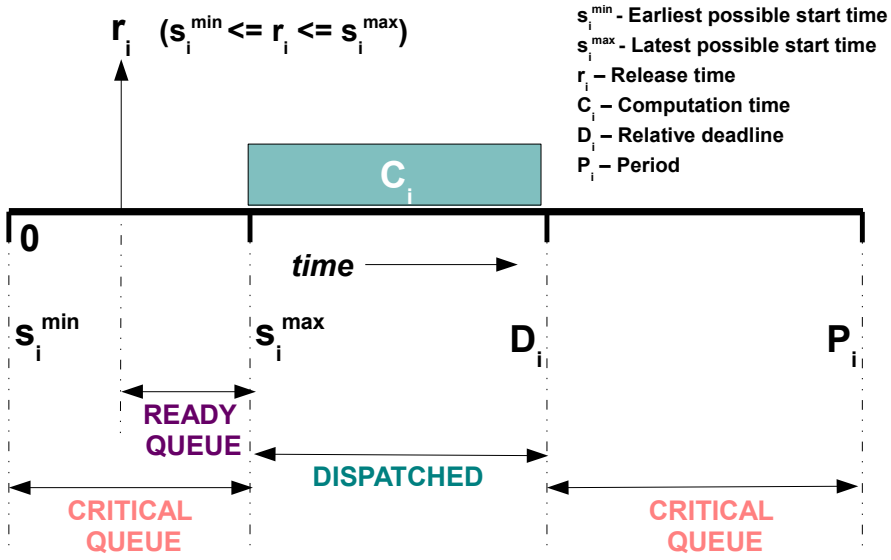
Temporal assumptions

In order to implement the CEDF algorithm, certain design considerations and assumptions were made in accordance with [30]. These are shown in figure .1. .1 (a) shows the temporal assumptions made on the ZASA task set. It is assumed that any task t_i, with a known period P_i, would have a fixed maximum computation time C_i, and a fixed relative deadline D_i. However, the release times may be periodic for some and non-periodic for others. This is denoted by r_i. For periodic tasks, the release time is fixed and always occurs at the beginning of the period. But the release times for some tasks, like t_2 (Data receive on UART), shown in table 8.1 of chapter 7, can be jittery. It can occur at any instance, within its period P_i. For tasks that exhibit release time jitter, D_i is assumed to be the end of the period. In other words, $D_i == P_i$. For periodic tasks, D_i can be either at or before the end of the period, i.e $D_i \leq P_i$.

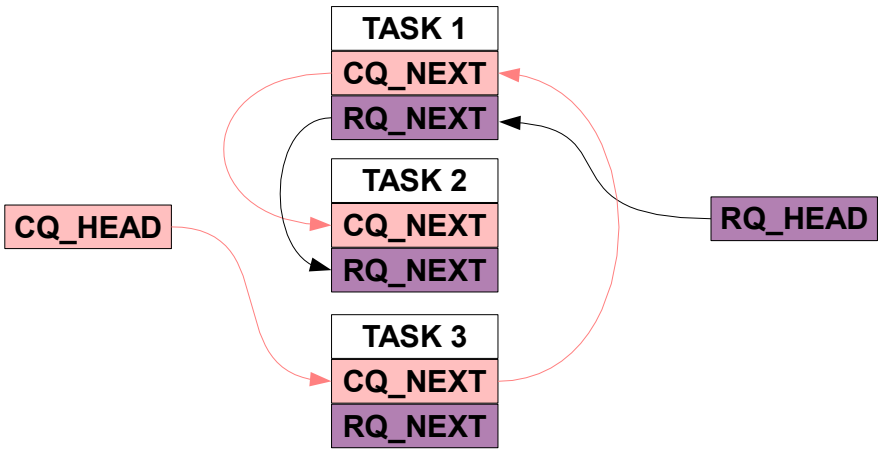
As per [30], there are two additional parameters that need to be considered for the efficient working of the CEDF algorithm. s_i^{min} is defined as the earliest possible start time of a task. It is generally equal to the beginning of the period, even for tasks with release time jitter. s_i^{max} [2] corresponds to the latest possible start time of the task, such that the task still meets its deadline. This means that $s_i^{max} = D_i - C_i$. The above equation derives the fact that starting the task at s_i^{max} will ensure that the task surely meets its deadline, where as starting the task after s_i^{max} may or may not result in the task meeting its deadline. Hence a task is never started after s_i^{max}. This also means that, the task can be released at or after s_i^{min}, but not after s_i^{max}, in any given period of the task. In other words $s_i^{min} \leq r_i \leq s_i^{max}$. If a task is released after s_i^{max}, it is assumed to be released at the beginning of the next period and executed in the same [3]. This assumption only applies to tasks with release time jitter, and is valid. This is because, C_i is a lot lesser than P_i and $D_i == P_i$. So that makes $(D_i - C_i)$ pretty small, compared to P_i. Postponing the release of the task by $(D_i - C_i)$ does not create serious repercussions on the normal functioning of the system.

[2] It is to be noted that absolute time is used for s_i^{max}, s_i^{max} and r_i, whereas relative time is used for all the other parameters

[3] This assumption is done only for the current implementation, on the MSP430. The CEDF paper does not advocate the same



(a) Temporal assumptions for CEDF



(b) Data structures for CEDF

Figure .1: The design considerations to implement CEDF

Data structures

[30] suggests the use of three main data structures to implement CEDF. This comprises of

1. *Ready Queue (RQ)* - containing list of ready/released tasks, which are ordered based on earliest absolute deadline first.

2. *Critical Queue (CQ)* - which contains all tasks and they are ordered based on smallest s^{max} first.

3. *Index structure* - containing information of all tasks in the system.

Since the number of tasks are less for ZASA, implementing an AVL tree is not required to maintain RQ and CQ, as done in [30]. Hence, RQ and CQ are implemented as singly linked lists, while index structure is an array of structures, with as many elements as the number of tasks in the system. Each element is used for storing all the task relevant information like period, deadline, computation time, etc., including two pointers *CQ_next* and *RQ_next*. These pointers are part of CQ and RQ respectively, and contain the index of the task element, which is next in queue, with respect to the current element. For example, if task t_i is ahead of task t_j in the ready queue, the RQ_next of t_i would contain the index of t_j and so on. If t_j is the last element in the ready queue, its RQ_next would be *INVALID*. This concept is depicted in Figure .1 (b). Two special pointers called *CQ_head* and *RQ_head* point to the task at the head of the critical and ready queues respectively. These two pointers are not part of the index structure. In .1 (b), t_3 is the head of the CQ followed by t_1 and t_2, where as t_1 is the head of RQ followed by t_2. t_3 is yet to be released and hence is not part of the RQ. If the queues are empty, CQ_head and RQ_head would be *INVALID*.

Working of the CEDF algorithm

.1 (a) also shows the status of a task t_i, at different times, within its given period. The task will be put into the critical queue at the beginning of the period with $s_i^{min} =$ *current absolute time*, and $s_i^{max} = D_i - C_i$. If the task is already in CQ, the s_i^{max} and s_i^{min} are updated to reflect latest times, if required. At time r_i, the task is released and hence, it is now inserted into the ready queue, depending on the value of its absolute deadline. At this point the task is part of both the CQ and RQ. When the scheduler dispatches the task

for execution at an appropriate time, it is removed from both CQ and RQ. So a task that is executing, is neither part of CQ nor RQ. After it finishes executing, its s_i^{max} and s_i^{min} are calculated anew, as applicable to the next period. The task is again inserted into CQ with updated values. Hence the task is only part of CQ but not RQ.

The code for CEDF was developed over a Run To Completion (RTC) scheduler platform. RTC is a simple algorithm that works on the concept of non-preemption, and ensures that once a task is dispatched for execution, it is not stopped mid way. The already existing code base of RTC was enhanced, to accommodate all the features required for CEDF. One of the important features of CEDF, is the postponement of tasks. The same scenario discussed in 9.2 with tasks, t_i and t_j, can be considered again for the current discussion.

The current implementation of the CEDF scheduler for ZASA, checks if all the following conditions are satisfied, before any task t_i, can be postponed, as described in [30] -

1. Executing t_i at the earliest possible start time would still result in a deadline miss for t_j.
2. Two different tasks are considered while doing the postpone check ($t_i \neq t_j$).
3. t_j has not missed its deadline already.

If all the three conditions are satisfied and t_i is to be postponed, then t_i is removed from both RQ and CQ first. Only if ($s_i^{min} + C_i$ s_i^{max}), then, t_i is reinserted only into CQ, with ($s_i^{min} + C_i$) as the key [4], instead of just s_i^{max}. Also, the release time r_i, and s_i^{min} are both updated as

$$r_i = s_i^{min} = s_j^{min} + C_j$$

If the task is not postponed, it is removed from both RQ and CQ, and dispatched for execution.

[4] However, the value of s_i^{max} remains as before and never changes to $s_i^{min} + C_i$.