Using Real-Time System Design Methods to Integrate SMPS Control Software with Application Software

Avik Juneja, Alexander G. Dean and Subhashish Bhattacharya
Dept. of Electrical and Computer Engineering
North Carolina State University
Raleigh, USA
ajuneja@ncsu.edu, agdean@ncsu.edu, sbhatta4@ncsu.edu

Abstract – A switch-mode power supply (SMPS) converts power efficiently between different voltage levels, making power optimizations through voltage scaling feasible. SMPS controllers are generally dedicated hardware (analog/digital circuits, microcontroller (MCU), digital signal processor (DSP)), and so are expensive to add to very low cost embedded applications.

In this work, we show how to integrate SMPS control software into the MCU running application software, which reduces system cost while increasing the design space and flexibility for developers. Real-time system design methods are employed to ensure SMPS voltage regulation quality, while retaining the original embedded application behavior. Our methods apply to a wide range of software task schedulers, from simple interrupt-based foreground/background systems to sophisticated preemptive real-time kernels (RTOS). We demonstrate our methods on a position-logging embedded system, with multiple voltage domains controlled in software, resulting in power savings.

Keywords—SMPS, DVS, DPM, RTOS, DC-DC, Power Supply, Embedded Systems, Firmware, Real-Time.

I. INTRODUCTION

Consider an embedded system consisting of an MCU and peripherals as shown in Figure 1. The operating voltage ranges of some sample components are depicted in Figure 2 with vertical bars. One approach is to power all peripherals at 3.3 V, as shown with the yellow horizontal line. However, we can save significant power by lowering the operating voltages, as $P \propto V^2$ for switching and resistive loads, and $P \propto V$ for constant current loads. For example, we can add two lower voltage domains (1.7V (blue), 2.7V (green)) to power the peripherals to operate at lower voltages.

Standard control approaches assume that the SMPS controller is implemented with digital logic or on a processor (MCU or DSP) entirely dedicated to that task [1] [2] [3] [4]. This additional hardware increases system cost and size.

We seek to reduce the cost of adding voltage domains by controlling their buck converters in software in the same MCU as the application software. To do this safely, we must ensure that the control software runs without timing interference from the application software. We use real-time (RT) system design techniques to tackle this challenge. A preemptive task scheduler and task priorities set by real-time design methods ensure proper responsiveness of all tasks without missing deadlines. This approach does not require allocation of dedicated CPU time (such as time slicing). Our contribution is a practical approach to integrate the SMPS control software into the application MCU while guaranteeing proper timing and operation of the SMPS control loops.

Figure 1: An embedded system with common peripherals

Figure 2: Operating voltage ranges of peripherals

A. Scope of Applicability

The embedded application design space and workloads cover a wide spectrum of power and performance. Depending on the platform used and the computational demands of the application, the CPU utilization can vary anywhere from below...
1% to as high has 100%. Similarly, the power variation between a performance-limited low-cost MCU and a high end floating point MCU with many peripherals can be quite significant, from as low as a few milliwatts to hundreds of watts, respectively. There is more variability in power vs. performance on these platforms for wide range of workloads, depending on software and hardware architecture, and optimizations, such as fixed point math, direct memory access (DMA), specialized functional units, ADC operation modes, variable scaling and optimizations, code reuse, modularity etc. Our methods provide a means to design and tailor an embedded application for maximum performance benefits with reduced power and span a wide range of applications, target hardware and software. We demonstrate a proof of concept using a sample embedded application with a variety of peripherals.

Our methods offer significant benefits to system designers, even when using an unexceptional buck converter without sophisticated circuit or control features. Performance can be improved further through synchronous switching (for efficiency) or feed-forward control (for response) and other methods. We leave these for future work.

B. Paper Overview
Section II introduces our design methods. Section III presents our experimental platform’s hardware and software. Section IV presents experiments and analysis to evaluate various aspects of the resulting system. Section V summarizes our contributions and presents future directions. Section VI discusses related work.

II. DESIGN METHODS

A. Overview

![Buck Converter Response to Step Increase in Load Current](image)

Figure 3. Buck converter response to step increase in load current

Our design method is based on understanding the voltage and time characteristics of a buck converter’s response to the worst-case combined load transient, shown in Figure 3. We evaluate the impact of delaying the execution of the control loop software, temporarily increasing the controller delay \( t_d \). We use real-time system design methods to determine the maximum interference timing interference (delay) possible from the application software. We then put these together to determine the impact of integrating the controller with the application software. We first present an overview of the proposed approach.

- Evaluate all of the loads on each supply rail to determine the maximum loading and unloading transients imposed by the rail’s peripherals.
- Design a buck converter using standard techniques, ensuring that \( V_{no} \) will not fall below the minimum acceptable voltage when using optimal control.
- Design a discrete time control law (compensator) for the buck converter.
- Use embedded and real-time system design methods to integrate the control loops into the application software. Leverage hardware peripherals for control operations, providing isolation from application software. Employ preemptive, prioritized task scheduling with interrupt service routines (ISRs).
- Use a schedulability test to verify all tasks and ISRs will meet their deadlines.
- Use real-time worst-case response time analysis methods to determine the maximum blocking time \( t_b \) based on duration of other ISRs and other non-interruptible code sections in the application.
- Derive an equation \( \Delta V_{\text{peak}}(t) \) to determine maximum output voltage transient, allowing analysis of impact of delayed execution of the control loop.
- Evaluate \( \Delta V_{\text{peak}}(t) \) to determine if voltage transient is acceptable. If so, design is complete. If not, adjust design, modify code to reduce \( t_b \), change scheduling approach. Repeat analysis.

B. Detailed Discussion

We now examine each of these steps in detail.

1. Evaluate Load Current Transients per Supply Rail
Each supply rail \( r \) powers peripherals in the set \( P(r) \). Determine the maximum worst-case loading transient for each supply rail \( r \) based on peripheral \( p \).

\[
\Delta I_{\text{max}}(r) = \sum_{p \in P(r)} \Delta I_{\text{loading, max}}(p)
\]

2. Design Buck Converter
Design a buck converter using standard techniques [5]. The buck converter shown in Figure 6 is targeted due to its ubiquity, simplicity, low cost and extensive analysis [6] [7] [8].

Select the components and operating parameters such that the maximum combined load transients \( \Delta I_{\text{loading}}(r) \) for supply rail \( r \) will not cause \( V_{no} \) to fall below the minimum acceptable voltage when using optimal control. A buck converter’s output voltage response to a step change in load current has been characterized [6] [8]:

\[
\Delta V_{out}(t) = -r_c \Delta I_{out} - t \frac{\Delta I_{out}}{C_{out}} + (t - t_d) m r_c + (t - t_d)^2 \frac{m}{2C_{out}}
\]

Here, \( m \) is the slope \( V_o/L \) of the inductor current. For loading transients \( V_o \) is \( (V_{in} - V_{no}) \); for unloading transients it is \( -V_{no} \). The delay from the load transient to the PWM duty cycle update is represented as \( t_d \).

The time of the peak output voltage deviation occurs after \( 1/4 \) of a period of the circuit’s natural frequency [7]. Adding in the ripple voltage may shift this time to one switching period:
\[ T_{\text{peak,OL}} = \frac{\pi \sqrt{L C_{\text{out}}}}{2} \pm T_{\text{sw}} \quad (3) \]

This time can be treated as the worst case (smallest) scenario, which will increase with inclusion of circuit parasitic values and resistances.

The size of the inductor relative to the critical inductance value (equation (4)) determines whether the output voltage will rise immediately or after an additional delay [6].

\[ L_{\text{crit}} = \frac{r_c C_{\text{out}} (V_{\text{in}} - V_{\text{out}})}{\Delta I_{\text{out}}} \quad (4) \]

A smaller inductance \((L < L_{\text{crit}})\) results in recovery of voltage immediately after \(t_d\).

\[ T_{\text{peak,CL}} = t_d \quad (5) \]

Equation (2) then reduces to:

\[ \Delta V_{\text{out}} = -r_c \Delta I_{\text{out}} - \frac{\Delta I_{\text{out}}}{C_{\text{out}}} \quad (6) \]

At that time, the output voltage reaches:

\[ \Delta V_{\text{peak,CL}} = -\frac{\Delta I_{\text{out}}}{C_{\text{out}}} (t_d + r_c) \quad (7) \]

However, with a larger inductance \((L > L_{\text{crit}})\) the peak output voltage deviation occurs later:

\[ T_{\text{peak,CL}} = \max\left\{ t_d, t_d + \frac{\Delta I_{\text{out}} L}{V_{\text{in}}} - r_c C_{\text{out}} \right\} \quad (8) \]

It is straightforward to apply this time to equation (2) to find \(\Delta V_{\text{peak,CL}}\) for the closed loop case with a large inductor.

3. Design Buck Converter Controller

We next derive the characteristic equation of the buck converter using state space analysis [5] [9]. This determines open-loop performance parameters such as rise time, settling time, over/undershoot and steady state error. The characteristic equation of the buck converter for a constant current load is approximately:

\[ P = 1 + (R_{\text{loss}} C + r_c C) s + LC s^2 \quad (9) \]

This includes the effective parasitic resistances for the non-synchronous buck converter \((R_{\text{out}} = r_L + r_d \epsilon D + r_y (1-D))\).

The compensator transfer function, \(D(z)\), is implemented as a discrete time difference equation using direct control law for simplicity [10]:

\[ d[n] = d[n - 1] + 0.2033 e[n] - 0.175 e[n - 1] \quad (10) \]

4. Implement SMPS Control

Next we use embedded and real-time system design methods to implement the control system for each SMPS.

First, design the system using the MCU’s hardware peripherals to isolate SMPS control operations from software scheduling and task blocking as much as possible. Use hardware timers to trigger the ADC sampling (at \(f_s\)) and the PWM signal generation (at \(f_{\text{PWM}}\)). Use hardware event signals to transfer data or control signals. Use ADC data update features or use direct memory access to transfer ADC results if possible to eliminate the need for ISRs to perform these operations.

Second, employ a preemptive task scheduling approach and leverage ISRs [11] [12] [13]. The task scheduling approach will dictate how to prioritize task execution at run time, eliminating the need for any time-slicing. Any control loop processing which remains after leveraging hardware peripherals should be executed in a high-priority ISR to minimize blocking. However, it is still vulnerable to blocking when higher-priority ISRs execute, if ISRs are not interruptible, or when ISRs are disabled for data atomicity or other reasons.

5. Verify Task Set Schedulability

The next step is to verify that the software task set is schedulable for a given scheduling approach and priority assignment. This means that no task will ever miss its deadline. This analysis starts by describing each software task and ISR in terms of its worst-case execution time, deadline, maximum release rate, and durations of critical sections (e.g. when interrupts or task switching may be disabled). Suitable tests are well-documented, including both exact response-time tests and inexact but safe utilization-bound tests [14] [15] [16] [17].

6. Find Maximum Blocking Time for SMPS Control Tasks

The SMPS control tasks will have been assigned high priorities. Use well-known methods to determine the maximum blocking time \(t_b\) for each SMPS control task [14] [17] [18]. These methods include blocking from interference from any higher priority tasks and non-preemptable code segments.

7. Derive Equation for \(\Delta V_{\text{peak}}(t_b)\)

Next we extend the load transient response analysis to consider the blocking time and discrete-time nature of the digital control system. Figure 5 shows the impact of blocking the
control loop execution for blocking times $t_{bq}$ and $t_{b2}$. The first time $t_{bq}$ is short enough that the control still has time to reduce $\Delta V_{\text{peak,CL}}$, compared with the open-loop case $\Delta V_{\text{peak,OL}}$. The second time $t_{b2}$ is long enough that the control responds too late and $\Delta V_{\text{peak,CL}} = \Delta V_{\text{peak,OL}}$.

The PWM output duty cycle is updated at a rate of $f_{sw}$, so the possible time delays $t_{bq}$ are quantized as an integer multiple of a switching period $T_{sw}$. We assume a one-period delay between ADC sampling and PWM duty cycle updating, but this delay could be reduced in future work.

$$t_{bq} = T_{sw} \left[ \frac{t_b}{T_{sw}} + 1 \right] \quad (11)$$

We can now determine the maximum voltage impact from blocking for time duration $t_{bq}$. There are two cases to consider. If $t_{bq} < T_{\text{peak,OL}}$, then control system will have an effect on the peak output voltage:

$$\Delta V_{\text{peak,CL}} = -\Delta I_{\text{out}} \left( \frac{t_{bq}}{C_{\text{out}}} + r_c \right) \quad (12)$$

If $t_{bq} > T_{\text{peak,OL}}$, then the control system’s response will occur after the open-loop peak and will not affect its magnitude:

$$\Delta V_{\text{peak,OL}} = -\Delta I_{\text{out}} \left( \frac{L}{C_{\text{out}}} \right) \quad (13)$$

8. Evaluate Resulting Voltage Transient

If output voltage transient $\Delta V_{\text{peak}}$ is acceptable, then this portion of the design process is complete. If not, the design may be adjusted in various ways.

a) Software Changes

Increasing the control loop frequency will reduce $t_d$ but will also proportionately increase the computational loading on the MCU. Reducing the blocking time will allow the control system to respond fast enough to reduce the peak voltage transient. This may involve raising the SMPS ISR priority, deferring work to lower priorities, making long ISRs interruptible, reducing critical section durations, not disabling all interrupts within critical sections, or adding a priority ceiling protocol.

b) Capacitor Changes

The “brute-force” approach is to increase the size of the output capacitor [7]. Increasing $C_{\text{out}}$ by a factor of $x$ will reduce the slope of the output voltage by a factor of $1/x$. It will delay the open-loop voltage transient by a factor of $\sqrt{x}$. Finally, it will also reduce the peak voltage by a factor of $1/\sqrt{x}$. However, this increases cost, volume and board area.

Using a capacitor with a lower ESR, $r_c$, will reduce the $\Delta V_{\text{peak,CL}}$ but also increase the $T_{\text{peak,CL}}$. For example, a tantalum capacitor has a much higher ESR than the ceramic. However, it will also reduce the critical inductance, making the system more likely to have a slower response even with optimal closed-loop control.

c) Inductor Changes

Increasing the inductance increases the delay until the open-loop peak voltage. For a closed-loop system with an inductor larger than the critical inductance, the larger inductor will increase the delay between the duty cycle update and the ramp-up of the output voltage. In both cases the magnitude of that peak voltage increases.

Reducing the inductance raises the amount of ripple voltage, unless the switching period is also decreased proportionally (which will increase power losses). The maximum inductor current will also increase, potentially raising inductor costs. This may also lead the converter to operate in discontinuous conduction mode, further complicating control and converter design.

d) Operating Voltage Setpoint

Raising the operating voltage will allow the system to tolerate a larger $\Delta V_{\text{peak}}$. However, this will also raise the system power dissipation.

III. SYSTEM DESCRIPTION

We demonstrate our methods on a prototype position-logging embedded system, shown in Figure 1. This features a GPS receiver, WiFi interface, LCD and microSD card. The system has three voltage domains (3.3 V, 2.7 V, 1.7 V) as shown in Figure 2, each powered by a software-controlled buck converter with 5V input. $f_{sw} = 50$ kHz and $f_c = 50$ kHz. The application MCU is a 16-bit Renesas RL78/G14 running at 32 MHz with 24 KB of RAM and 256 KB of flash ROM [19] [20].

It can operate from 1.6 to 5.5 V. As with many other inexpensive embedded MCUs, there is no hardware support for floating point math. Each of the power supply rails is driven by a simple non-synchronous buck converter, shown in Figure 6. The SMPS control software was implemented in the RL78 MCU using a standard control law (eqn. (10)) derived using digital control theory for a PI compensator [1] [2] [5] [8] [21] [22].

![Figure 6: Circuit diagram and prototype buck converters](image-url)

The overall control sequence is outlined in Figure 7 and is implemented with both hardware peripherals and software (in ADC_ISR). The control task was implemented in a high-priority ISR. We performed various optimizations to reduce the computation time. Major benefits came from using fixed point math, and eliminating many ISRs by using the Event Link Controller to allow peripherals to trigger each other. We reduced the computation time $t_{control}$ for one SMPS down from over 50 µs down to only 2.24 µs with the MCU operating at...
32MHz. For three domains the total computation time is 5.56µs. Thus, running these domains consumes only 27.8% of the MCU’s time, leaving most of it free for application processing.

B. Timing Interference Analysis

We measured the impact of timing interference which delays (blocks) the execution of the SMPS control ISR, delaying the response to a load transient. Figure 10 shows the SMPS response (with optimal control) to a step load transient with increasing blocking times.

<table>
<thead>
<tr>
<th>Normal Sample</th>
<th>JIT Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Sample</td>
<td>JIT Sample</td>
</tr>
</tbody>
</table>

Figure 11: Voltage deviation under normal and JIT sampling technique for various blocking times

C. Energy Efficiency Analysis

Burst mode operation was added to improve buck converter efficiency by reducing switching losses when operating under very low load conditions.

Figure 12 shows that when the duty cycle falls below a threshold, the controller disables SMPS switching operations so the load circuit is powered by the output capacitor. When the output voltage falls below a threshold, the control software enables SMPS switching operations, allowing the system to charge the output capacitor again.
Figure 12: Burst mode implementation of SMPS under very low duty cycle (small loads)

Figure 13 shows how converter efficiency improves significantly at lower load currents.

Figure 13: Efficiency improvement due to burst mode operation in light load condition

Figure 14: Increased efficiency of SMPS at small loads using burst mode implementation

The overall efficiency curve appears in Figure 14 and Figure 15. If better efficiency is needed at higher loads, the converter can be changed to a synchronous buck converter to reduce the conductive losses.

Figure 15: Buck converter efficiency under various load conditions

D. Output Capacitor Selection

We evaluated the transient response of the buck converter using both ceramic and tantalum output capacitors, as shown in Figure 16. Ceramic capacitors are more expensive but have a lower ESR which reduces ripple voltage significantly. However, this reduces the critical inductance, leading to a slower response to transients even with optimal control [7] [8].

Figure 16: SMPS response to load transient with different output capacitor technologies

E. System Energy Use

Three 1F ultracapacitors were employed to power the system. The total energy available is constant for all test cases. The ultracapacitors were charged for a fixed amount of time to voltage \( V_{\text{init}} \) and then allowed to discharge by powering the system. The amount of time, \( t_{\text{discharge}} \), taken by the system to drop the voltage of the ultracapacitors to \( V_{\text{final}} \) determines the rate of energy consumption or the average power, \( P_{\text{avg}} \) consumed by the system, given as:

\[
P_{\text{avg}} = \frac{E_{\text{total}}}{t_{\text{discharge}}} = \frac{1}{2} C \frac{V_{\text{init}}^2 - V_{\text{final}}^2}{t_{\text{discharge}}}
\]  

(14)

The average power reduction in various operating modes of the embedded application is normalized and depicted in Figure 17 and Figure 18. It can be clearly seen that using an SMPS with single voltage domain over a linear regulator results in power savings of up to 59%. Using multiple domains results in power savings of 66%. When comparing the use of single vs. multiple voltage rail SMPS design, up to 26% power savings can be observed, as depicted in Figure 18. Although these
power gains are modest, they were achieved with three non-synchronous buck converters with limited efficiency. Converting them to synchronous converters would improve efficiency significantly with negligible software impact. Furthermore, load balancing on supply rails can significantly improve the average power consumption in all modes of operation. In the current system, most of the load is on the 3.3V rail (89.15%), resulting in little improvement of multi-rail over single rail SMPS. Depending on low power applications, the load can be distributed to 1.7V and 2.7V rails.

Due to the tight coupling of the hardware and software in this work, there are limitations which need to be considered during the design process. These include:

- SMPS control software must be robust and isolated from application software to avoid faults such as memory overflow, peripheral misuse, etc.
- Computational loading of the control software will lead to application scheduling constraints.
- Development time will be increased.
- Sub-optimal load distribution may result in limited energy savings.

Future work exists in various directions. Processor computational loading can be reduced by leveraging additional hardware peripherals (e.g. DMA controller), applying dead-band control, and hardware-assisted feed-forward control. Transient response can be improved using feed-forward control. Cost and size can be reduced by increasing switching frequency. Although this present work examines robustness of the control software with respect to timing interference by the application software, the SMPS controller’s peripherals and memory space must also be protected from interference.

VI. RELATED WORK

In the power supply design community, digital control of switching converters has been a very active field [1] [10] [21] [22] [23] [24]. The transient response has been investigated to identify critical relationships between circuit parameters [7] [25]. It is especially important for voltage regulator modules which must handle the large transients of power-hungry high-performance processors [6] [8].

There have been many investigations into the software issues with controlling power converters using MCUs and DSPs [2] [3] [4]. In fact, many MCU and DSP vendors provide working reference designs for such applications. The real-time design community has investigated the implications of jitter on digital controllers for other applications and suggested various solutions [14] [26] [27].

Designers of high-performance processors (e.g. personal computers and cellphones) have long relied on buck converters for dynamic voltage scaling. There are many papers which concentrate on power management at the microprocessor level [28] [29] [30] [31] [32]. These discuss various techniques to avoid voltage fluctuation by use of sense or prediction techniques, such as application interactions, microprocessor architectural changes, hybrid multi-phase converters, etc.

VII. REFERENCES


