ASSIGNMENT: SWITCHED MODE POWER CONVERTER SIMULATOR/EMULATOR

BL40A1101 Embedded System Programming

Lappeenranta–Lahti University of Technology LUT

LUT SCHOOL OF ENERGY SYSTEMS

2022

Danila Goncharenko

Aleksandar Ivanov

Samuli Pietarinen

Mikael Forss

Table of contents

1 Introduction

The group has constructed a DC- converter controller and a converter model that is controlled with the interface in Zybo Z7-10 development board.

The video of the simulation was attached to the submission in assignment return folder. It is also available on youtube: https://www.youtube.com/watch?v=vxCHkgKxCbo

The source code is available in github: https://github.com/Smiche/dc-converter-zybo

2 C coding

2.1 Main

The code is a C program for an embedded system that utilizes the FreeRTOS operating system. It is designed to control some aspects of the system, involving the generation and modulation of a signal, and involving feedback control using a PID controller.

The program consists of several tasks, which are essentially independent units of execution that can be scheduled to run concurrently. One task, called “Idle”, is responsible for performing idle processing, so the system does not modulate (converter is off).

Another task, called “StateControl”, puts the system to the configuration mode, in which the second button allows user to select either the Kp, Ki, and Kd parameters, and the third and fourth buttons allow user to decrease or increase the parameter value or the reference output voltage in the modulating mode. “StateControl” task is responsible for the handling of inputs and switching between the different states of the controller. Configuring of the controller parameters is done within the same task. If the mode changes to Modulating or Idle, the StateControl task is responsible for starting tasks that are responsible for these modes.

“Modulating” task is responsible for the controller modulation. It controls the output voltage of the model. When the mode changes, “Modulating” and “Idle” tasks are responsible for exiting their loops and terminating within a second.

Resources used concurrently are protected using Semaphores, Queues and atomic operations. Mode changing, modulation and pid configuration use semaphores, while inputs from switches and buttons are handled with queues.

In addition to these tasks, the program also includes several other functions that are used to handle input and configuration. These functions include “vUARTCommandConsoleStart”, which listens for commands from a UART, and “init_inputs”, which initializes the input switches and buttons of the embedded system.

The program also includes data structures for storing configuration information. One such data structure is called “pidConfig”, which stores configuration information for the PID

controller. Another is called “modulationConfig”, which stores configuration information for the modulation signal. These data structures are guarded by semaphores, which are used to synchronize access to shared resources and prevent race conditions.

2.2 FreeRTOS, console, serial

The FreeRTOS is being used to manage and schedule the tasks that are needed to implement the DC converter. The tasks that are responsible for reading input voltage and current values, tasks that control the switching of the power switches, and tasks that handle communication with other devices (e.g., through a serial interface).

By using FreeRTOS, the tasks can be scheduled to run at the appropriate times and share resources (such as memory and hardware peripherals) in a controlled manner. This can help to ensure that the DC converter operates in a predictable and reliable way, even in the presence of real-time constraints and interruptions.

The "FreeRTOS_CLI", "serial", "UARTCommandConsole", and "xilinx.spec" files were downloaded from Open Source for the given application.

2.3 Inputs

The code initializes and defines a task that continuously polls the status of a set of switches and buttons connected to a Zybo Borad using Xilinx. The task reads the current values of the switches and buttons using the XGpio_DiscreteRead function from the Xilinx GPIO library and stores them in variables sw_values and bt_values, respectively. It then compares these values to the previous values stored in the variables last_sw_values and last_bt_values. If the values have changed, the task updates a struct called input_statuses with the new switch and button values and sends the struct to a queue called inputs_status_queue. The task then saves the new values in last_sw_values and last_bt_values for comparison in the next iteration. The task repeats this process indefinitely, with a delay defined by the pollDelay variable between each iteration. The queue ensures that the button states are passed safely onto the StateControl task.

2.4 LEDs

Three timer counter devices are coded for use in driving an RGB LED. The devices are initialized and configured using the XTtcPs_Config and XTtcPs_LookupConfig functions, which use the device IDs provided in the zynq_registers.h header file to find the corresponding configurations in the device tree.

Once the devices are initialized and configured, the function sets options, calculates the interval and pre-scaler values based on the desired output frequency of the timer counter devices, and sets the interval and pre-scaler values for each device. The function also sets the match values for each device to zero, which keeps the LED initially turned off. Finally, the function starts the timer counter devices.

The timer counter devices are then used to modulate the pulse width of the red, green, and blue channels of the LED, with the duty cycle of each channel controlled by the match value for the corresponding timer counter device. When a match value is set for a timer counter device, the device will output a pulse with a width equal to the time between the current timer counter value and the match value. By changing the match values for each timer counter device, the duty cycle of the corresponding LED channel can be changed, resulting in changes to the overall color of the LED.

2.5 PID controller and Converter

A PID controller is a feedback control system that is widely used to maintain a process variable "voltage" (U3) at a desired setpoint "Voltageref" by continuously adjusting an output based on the difference between the setpoint and the error. The output of the PID controller is calculated as the sum of three terms: the proportional term (yp), the integral term (yi), and the derivative term (yd).

The function takes several input arguments from buttons and the UART interface: Kp, Ki, and Kd that are the gain constants of the proportional, integral, and derivative terms, respectively. The controller's output serves as a reference for the pulse width modulation (PWM) output of a timer.

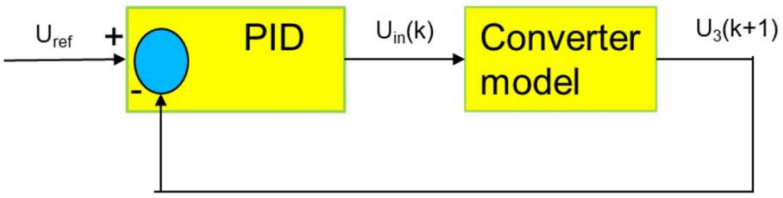

Figure 1. discrete PID controller (Tuomo Lindh)

"Saturation_limit" is a limit on the integral term, which helps to prevent windup, a condition in which the integral term becomes too large and causes the output to become unstable. If integral term exceeds the saturation limit, the integral term is set to the previous value

The function calculates the error by subtracting the current process variable from the setpoint. It then calculates the proportional, integral, and derivative terms of the PID controller based on the error and the gains. The sum of those is sent to converter model.

Converter model mathematically calculates the voltage $u_3(k+h)$ using the discretized equation shown on the figure 2, coded in C language. At the beginning values of $i_{1\text{-}3}(k)$ and $u_{1\text{-}3}(k)$ are set to be 0. The model continuously updates these values by replacing old values with newly calculated ones. The voltage $u_{in}(k)$ is the output from PID controller that is also constantly changing over time.

$$\begin{bmatrix} i_1(k+h) \\ u_1(k+h) \\ i_2(k+h) \\ u_2(k+h) \\ i_3(k+h) \\ u_3(k+h) \end{bmatrix} = \begin{bmatrix} 0.9652 & -0.0172 & 0.0057 & -0.0058 & 0.0052 & -0.0251 \\ 0.7732 & 0.1252 & 0.2315 & 0.07 & 0.1282 & 0.7754 \\ 0.8278 & -0.7522 & -0.0956 & 0.3299 & -0.4855 & 0.3915 \\ 0.9948 & 0.2655 & -0.3848 & 0.4212 & 0.3927 & 0.2899 \\ 0.7648 & -0.4165 & -0.4855 & -0.3366 & -0.0986 & 0.7281 \\ 1.1056 & 0.7587 & 0.1179 & 0.0748 & -0.2192 & 0.1491 \end{bmatrix} \begin{bmatrix} i_1(k) \\ u_1(k) \\ i_2(k) \\ u_2(k) \\ i_3(k) \\ u_3(k) \end{bmatrix} + \begin{bmatrix} 0.0471 \\ 0.0377 \\ 0.0404 \\ 0.0485 \\ 0.0373 \\ 0.0539 \end{bmatrix} u_{in}(k)$$

Figure 2. discretized equation for the converter (Tuomo Lindh)

The LED light brightness demonstrates how big the voltage value is by shing brighter.

3 Results

The program was tested three times, where the system was controlled using different methods. First time with the defaults of PID controller (Figure 3). Second time with changing PID controller parameters with buttons (Figure 4, 5). Last time with using console commands (Figure 6) and buttons (Figure 7). SW0 toggled the printing of the reference & actual voltage for the chart plotter. The PID saturated to the exact value over time, and all input methods worked. The video mentioned in the Introduction shows the simulation in detail.

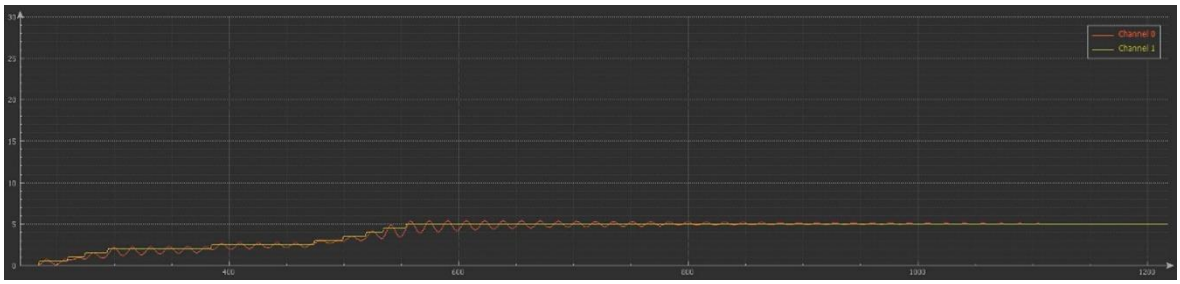

Figure 3. defaults of PID controller (Aleksandar Ivanov)

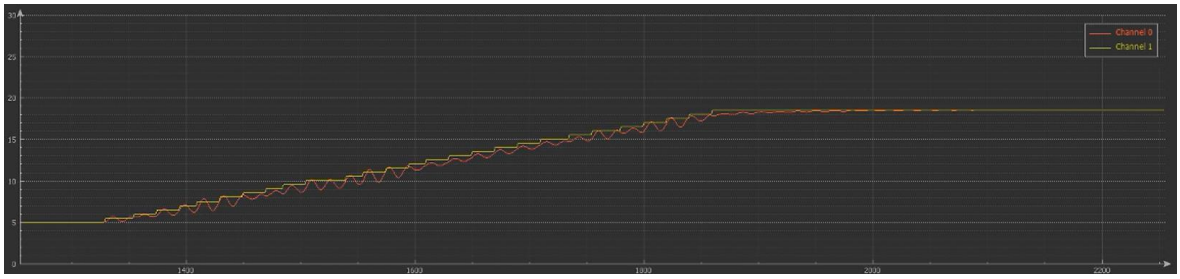

Figure 4. increasing parameters with buttons (Aleksandar Ivanov)

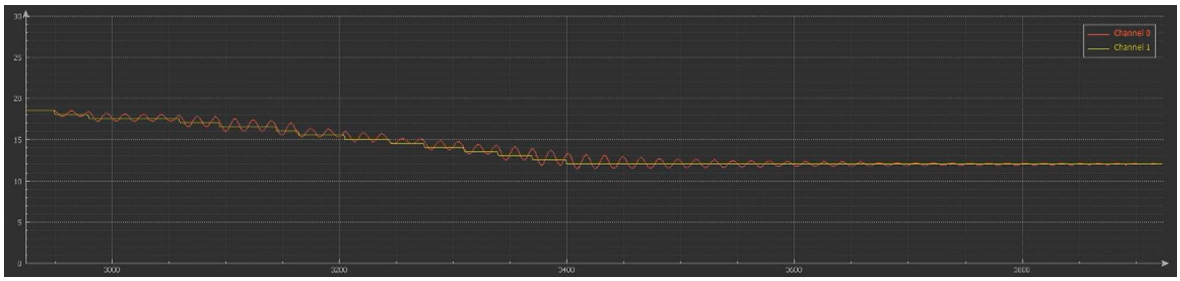

Figure 5. decreasing parameters with buttons (Aleksandar Ivanov)

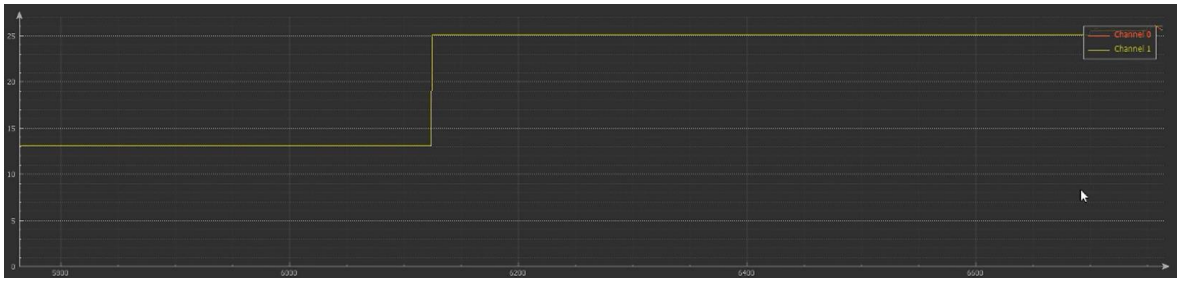

Figure 6. using console commands (Aleksandar Ivanov)

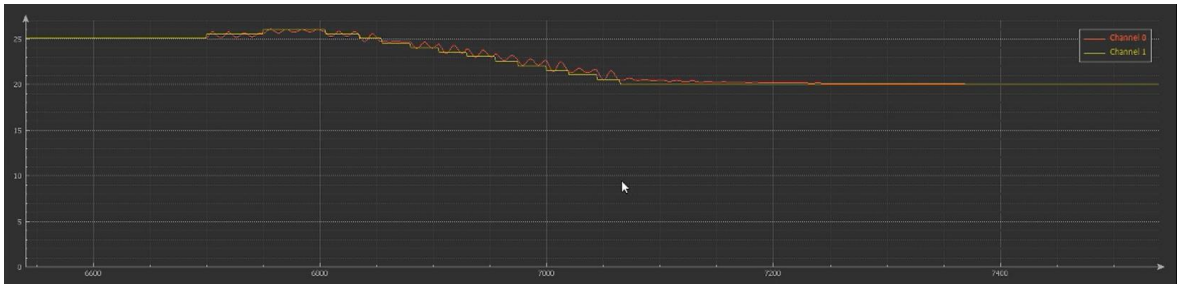

Figure 7. using console commands and buttons (Aleksandar Ivanov)

As the result of this assignment, students gained experience utilizing the FreeRTOS model in their code, implementing a PI(D) controller, converting mathematical equations into C code, and working in a team to utilize the Zybo Z7-10 development board environment to complete all the given tasks.