Assignment: Switched mode power converter simulator/emulator

Assignment first due date: 31.1.2025

The assignment will be graded in conjunction with the nearest Moodle examination grading.

Working method: The work will be done in groups of three to four people. The students can form the groups by themselves. Please use the group formation tool, that's located in the Moodle page's Assignment-tab. You can look for teammates in the course's discussion forum; course personnel help in forming the groups, if necessary. Groups are encouraged to help each other. However, copy/pasting other groups code is forbidden. Lecture examples can be used where applicable. The group submits their work in Moodle. The material returned includes a short description of the program and source code. A short video showing the testing of the program is appreciated.

The work consists of a basic version and four different bonus tasks, which provide extra points.

Evaluation: Accepted work 1-100 points + optional bonus tasks 0-10 points / bonus task (max. 2 tasks graded =0-20 points).

Work content:

Power converters are driven by DSPs, FPGAs or other high-performance processors. In development phase, these devices are often emulated by using electronics that describe the functionality and dynamics of a real device. Doing that, the developer can construct a control system without a ready-made device and load. In this work, we use a simulator (plant model) that can be controlled by your control software.

Your task is to construct a DC- converter controller and a converter model that you control with your controller. DC- converter is the basic form but you get bonus points if you are able to run the system to emulate 50-Hz inverter.

This exercise work was designed so that it doesn't require extra hardware besides a Zynq board and a computer. If you're working with the Nucleo board, you may need to use external LEDs for multiple PWM outputs and external switches for a multi-switch user interface. Some of the terms and concepts may not be familiar to you at this point. These terms are marked as green and examples will be given during lectures.

The program should include the following:

- Real-time program structure. The simplest one applicable is a Polling with interrupts system. Your own simple scheduler (lectures 5 and 6) is another possibility, and a real-time operating system (FreeRTOS, lecture 10) is the third option. (Notice that if you use FreeRTOS on the Zynq board, use only tasks, no own interrupts; there is no straightforward way to have these working with Zynq. On the Nucleo board interrupts with FreeRTOS should work fine but remember to use critical sections).
- Converter model. This model is a simple state space model given to you in eq. 3.
- Controller. This is simply a PI-controller (lecture 7).
- PWM output constructed from controller output driving an RGB-led (led brightness as a visualization of the controller output voltage / input voltage of the converter model).
- User interface that comprises of buttons, LEDs and UART communication using the serial terminal.

Specifically, to Groups working with Nucleo board.

- If no additional components are not available for your case. You can create your own solution, for example, like in the following:

  Button can change only the mode (idle->modulation etc), but if UART takes the semaphore, buttons should not interrupt. Every other functionality besides the mode change is done via UART. (Additionally, buttons timed semaphore can be used, when button change the mode, UART cannot change the mode for some time but probably can work within the mode).

Some other general tips:

- **Do not use delay functions like sleep or waiting loops in your code outside the initialization phase (e.g. setting up the device). This does not include functions like "vTaskDelay" or "vTaskDelayUntil" that FreeRTOS has, as those only block the execution of individual tasks, not the entire program.** While these delay functions can be used when testing things (like in the exercises), they are not suitable for real-time programs. In hard real time systems, using delays may prevent critical operations such as control and it is also hard to guarantee that the delay will be of the desired length.
- The lecture 4 page contains example code for different interrupts like UART and buttons/switches. These can provide a good foundation for building your own code.

The control part

The control is done by a discrete PI(D) controller.

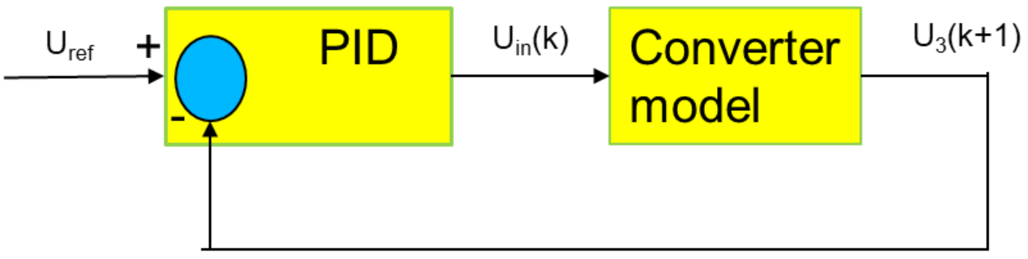

Fig. 1 Control schema.

The PI-controller has P- and I-terms (Kp, Ki) that you change using buttons and UART (e.g. Xilinx console). The controller's output is given as a reference to PWM output of a timer. In the basic version, the converter module takes a reference directly from the PI-controller output (H-bridge taken as ideal).

The converter is modeled as a discrete state-space model (matrix calculation) that is given to you in eq. 3.

User interface comprises of serial terminal (e.g. "SDK Terminal") console, buttons and LEDs. You have to implement three modes of operation. The first button selects mode between 1. configuration,

2. idling and 3. modulating modes, the second button in configuration mode selects configurable parameter Kp or Ki, the third button decreases value: reference output voltage in modulating mode or parameter value in configuration mode. The fourth button increases values correspondently. In idling mode, the system does not modulate (converter is off) and in modulating mode controller controls the output voltage of model. The mode changes are indicated by LEDs and by writing to console. The console can request mode changes and set parameter values or reference voltage depending on the current mode.

If serial console requests the configuration mode, it must set a binary semaphore and release it when it leaves the configuration mode. Buttons should not do anything when semaphore is reserved, **On the Zynq board it's currently not possible to disable button interrupts**, so they don't have to be disabled there. On the Nucleo board, you can enable/disable the relevant IRQ when necessary. Also, when buttons are used for any changes, another binary semaphore could prevent changing the mode or values from UART for at least 5 seconds (see evaluation matrix). For both cases, a timer semaphore could be used. If timer semaphore is too difficult, use a passing procedure semaphore that's released when leaving the configuration mode. If you use non-preemptive scheduler, use passing procedure semaphores. Waiting semaphores in general require a great deal of attention by the developer to implement successfully, so use them only if you use FreeRTOS and use MUTEX type of semaphore (e.g. xSemaphoreCreateMutex(void)). Reason being that it's easy to lock the system into a perpetual waiting state with nothing happening when using waiting semaphores.

If you print out the measurement value of output voltage for example, send the value slowly enough (0.1s...1s between) so that console buffer doesn't overflow (and clear console every now and then).

The modeling part (You do not need to do any modeling in this assignment, but use the discrete version matrix and vector calculations, this is just for you to understand how the model was created).

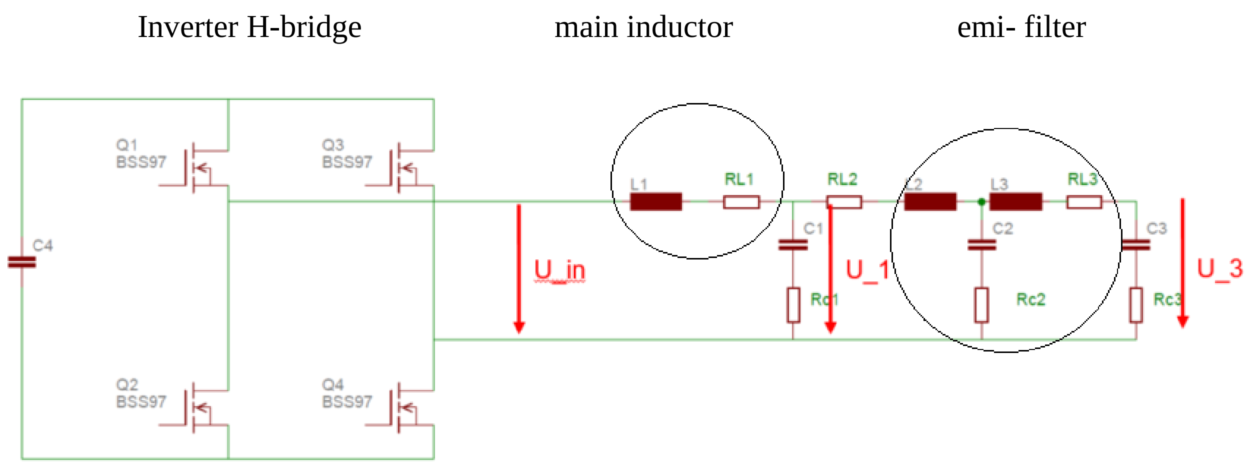

Fig. 2 Converter/Inverter schema.

State variables are: Voltages over capacitors and currents over inductors. The model in continuous time is the following. Notice that on the left-hand side of the equal sign we have derivatives of states which are not very well printed from equation editor.

$$\begin{bmatrix} \dot{i_1} \\ \dot{u_1} \\ \dot{i_2} \\ \dot{u_2} \\ \dot{i_3} \\ \dot{u_3} \end{bmatrix} = \begin{bmatrix} -\frac{R_{c1}}{L_1}-\frac{R_{L1}}{L_1} & -\frac{1}{L_1} & -\frac{R_{c1}}{L_1} & 0 & 0 & 0 \\ \frac{1}{C_1} & 0 & -\frac{1}{C_1} & 0 & 0 & 0 \\ \frac{R_{c1}}{L_2} & \frac{1}{L_2} & -\frac{R_{c1}}{L_2}-\frac{R_{c2}}{L_2}-\frac{R_{L_2}}{L_2} & -\frac{1}{L_2} & \frac{R_{c2}}{L_2} & 0 \\ 0 & 0 & \frac{1}{C_2} & 0 & -\frac{1}{C_2} & 0 \\ 0 & 0 & \frac{R_{c2}}{L_3} & \frac{1}{L_3} & -\frac{R_{c2}}{L_3}-\frac{R_{c3}}{L_3}-\frac{R_{L_3}}{L_3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{C_3} & 0 \end{bmatrix} \begin{bmatrix} i_1 \\ u_1 \\ i_2 \\ u_2 \\ i_3 \\ u_3 \end{bmatrix} + \begin{bmatrix} {}^{1}/{L_1} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u_{in} \quad (1)$$

The output of model is output voltage u_3.

$$y = Cx = [0\ 0\ 0\ 0\ 0\ 1]\begin{bmatrix} i_1 \\ u_1 \\ i_2 \\ u_2 \\ i_3 \\ u_3 \end{bmatrix} \tag{2}$$

D= [0] as usual.

Equation 1 can be discretized using different methods (see lecture notes). In this work we used MATLAB’s c2d-function with ‘impulse’ option (this year). Discretization was calculated using 50 kHz sampling rate. So, this should be the execution interval of the model if we want to simulate a process in real-time. However, both controller and model can (or should) be executed using a much longer sampling time, so that you can visually see what is happening.

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

Output $u_{out} = Cx = [0\ 0\ 0\ 0\ 0\ 1]\begin{bmatrix} i_1(k+h) \\ u_1(k+h) \\ i_2(k+h) \\ u_2(k+h) \\ i_3(k+h) \\ u_3(k+h) \end{bmatrix}$ (3)

BONUS 1: Modification of converter to inverter simulation:

Your task is to construct a single-phase inverter simulation. The inverter model is same as converter model (using eq. 3). The inverter should produce inverter output voltage *U*_out using a sinusoidal 50 Hz signal with an amplitude that can be constant as *U*_in. (see Fig. 1)

BONUS 2: Feed PWM modulated signal to the simulation model

Your task is to construct a H-bridge simulation. The PI-controller's output is given as reference to the Timer/PWM-module. PWM output is fed to the inverter model and LED. In order to produce the modulated value (PWM output) to run the model, you should construct a counter match ISR where you then output a positive or negative value (zero if negative seems too complicated).

BONUS 3: MQTT to a database (**Zynq only**)

Boot Linux on the Zybo board and use MQTT to send data to another computer on the network and store it in a database (this MQTT one also uploads information to ThingSpeak). This will be an entirely separate thing from the preceding tasks and there are two paths for you to choose from:

- Launching Ubuntu on the Zybo, and sending sine function values produced in Ubuntu to the database
- Launching an OpenAMP Linux system on the Zybo and sending data produced in the bare metal to the database (the state of a switch e.g.)

Doing both above doesn't provide you with bonus marks, since MQTT is the main thing. There are ready-made images available for both Ubuntu and OpenAMP and instructions to cover the whole process in either case. OpenAMP is more complex, since you need to launch a bare metal and Linux program in addition to the MQTT client.

This and bonus 4 will largely be a demonstration for students to familiarize themselves with IoT. The available software environment (Linux/Windows) largely influences the number of problems that might arise during the process. Students will be provided with the necessary files for booting Linux on Zybo and the necessary source code. Some steps still require the programs to be compiled.

Showcase your system on video and return it. On the video, you need to show the MQTT client running on Zybo and sending data to the broker (either on Zybo's own screen or a serial terminal). The video also needs to show the broker receiving data, the Python script updating the database and finally the contents of the database as instructed in the guide. Showing the ThingSpeak portion is not required.

BONUS 4: OPC UA to a database (read the description of bonus 3 for details) (**Zynq only**)

Boot Linux on the Zybo board and use OPC UA to send data to another computer on the network and store it in a database. This has largely the same type of procedure as bonus 3, except that this procedure won't be uploading information to ThingSpeak. You can choose between Ubuntu and OpenAMP in this case as well.

Showcase your system on video and return it. On the video, you need to show the OPC server running on Zybo and sending data to the other computer (either on Zybo's own screen or a serial terminal). The video also needs to show the Python script receiving and updating the database and finally the contents of the database as instructed in the guide.

Evaluation

Feature	**Minimum(50-65 points)**	**Fair(66-80 points)**	**Good(>80 points)**
Functionality	Control and model works (almost)	Control and model works (fairly)	Control and model works well
UART	communication works	shows clearly, allows changes of parameters and reference voltage	handles state changes
Code style	Readable code.	Well commented code.	Well commented code and self-explanatory code.
Program structure	Working code	Structured parts Initialization phase(s)	Modular code, Separate files, Information hiding done, Interface functions, Synchronization of tasks
Scheduler	Polling with interrupts without any scheduling structure. **NOTE: Using delay functions (like sleep) outside the initialization phase also results in minimum marks for this category. This does not include functions like "vTaskDelay" or "vTaskDelayUntil" that FreeRTOS has.**	Polling with interrupts with some scheduling structure.	Scheduled tasks (own implementation, or Free RTOS): Systematic Prioritized Clear Implementation
Protection of shared data	None	Semaphores almost correctly used	Semaphores constructed correctly. Either passing semaphores or timed semaphores, which release after a specified time if not updated. (Or waiting semaphores, if you're brave enough.)
Controller	Working PI. **NOTE: THE CONTROLLER'S PERFORMANCE OR STABILITY WILL NOT BE EVALUATED.**	Saturation Proper controller structure. Reentrant controller function	+ Extra features e.g. Anti-winding or filtering derivative parts or otherwise more complex control algorithms
User interface	Buttons and LEDs working	Buttons/console implementation, fair console menu structure, Data protection included	Good menu structure, responsive buttons/console implementation, data protection included
Report	Code only	+ Short description of work and functionality implemented	+ Video showing testing. (Nice if the video shows board and running code simultaneously. Audio commentary or subtitles would be nice to include as well. Also, good to include the use of semaphores).
Delivery date	30.4.2025. Code returned to Moodle (absolute deadline for this year's course; you can however retake the course next year).	28.2.2025. Code returned to Moodle.	31.1.2025. Code returned to Moodle.