Project – Voltage Regulating Relay

Voltage regulating relay overview. Functional description. PC interface. Software development.

Introduction

A voltage regulating relay (VRR) is a power system device that initiates the changing of taps on a transformer to regulate the supply voltage under changing load conditions. A VRR is better than an electromechanical relay because it is able to have completely customisable settings and is able to be set and interrogated by a Supervisory Control and Data Acquisition (SCADA) system.

Since large power systems utilise a 3-phase circuit, there are three voltages (labelled v_a, v_b and v_c) that require independent monitoring. The aim of this project is to implement a 3-phase VRR that is able to be set and interrogated remotely (via the PC and USB interface).

VRR Overview

The VRR is a real-time system. It needs to retrieve waveform amplitude information (a "sample") from an analog-to-digital converter at precise intervals, do calculations with those samples, and perform timing operations in the event of an over-voltage or under-voltage situation. It needs to start operating instantly upon startup – thus a PC is unsuitable. It is relatively easy to develop a stand-alone real-time embedded system.

P.2

We will therefore exploit the GUI of Windows® and the real-time capability of a microcontroller to develop the VRR, as shown below:

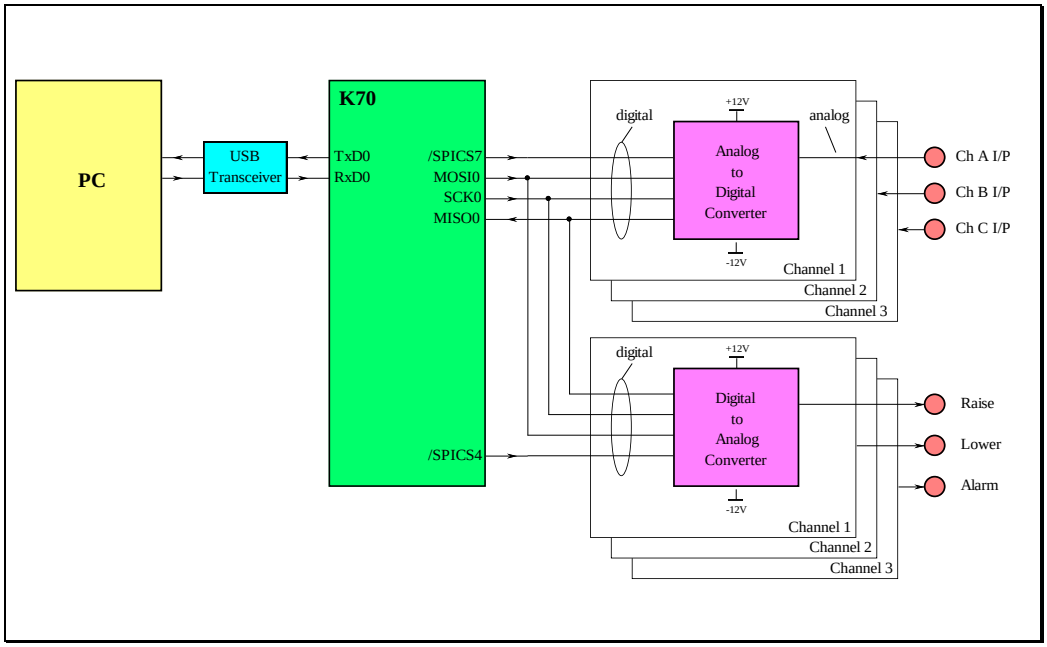

Figure P.1 – VRR Block Diagram

The PC is responsible for providing a user interface for the VRR.

A USB interface is the primary means of communication with the VRR. One of the six available UARTs on the μC is used for this purpose (UART2). The operation of the VRR is controlled and monitored from a PC using the Tower protocol.

The μC is responsible for communications with the PC and for retrieving information about 3 independent voltages from the external analog-to-digital converter (ADC) via a Serial Peripheral Interface (SPI). The input voltages are assumed to have been derived from the voltages in the power system by additional signal conditioning circuitry which is not shown.

The μC is also responsible for generating the so-called “Raise” and “Lower” signals which are used to initiate a tap change on the transformer, thus preventing both undervoltage and overvoltage conditions on the power system. It will also generate an “Alarm” signal which indicates that the power system’s voltage is out of allowed bounds and a tap change will commence after the set timeout period. These are outputs generated by the digital-to-analog converter (DAC) via the SPI.

The ADC circuitry converts the voltages appearing on the external connectors from an analog signal into a digital value. The DAC circuitry converts a digital output value into an analog voltage which appears on an external connector.

Both the ADC and the DAC are 16-bit, bipolar, with a range of ±10 V.

Functional Description

Functional Requirements

The VRR must meet the following functional requirements:

Specification	**Value**
Phases (channels)	3
Voltage Range	0 to 5 V RMS
Nominal Voltage	2.5 V RMS
Voltage Limits	2 V RMS (low), 3 V RMS (high)
Frequency Range	47.5 Hz to 52.5 Hz
Timing	Definite (5 seconds), and Inverse
Accuracy	5% (voltage and time)
Measurement Method	True RMS
Sample Period	16 samples per cycle minimum

Example of Voltage Regulation

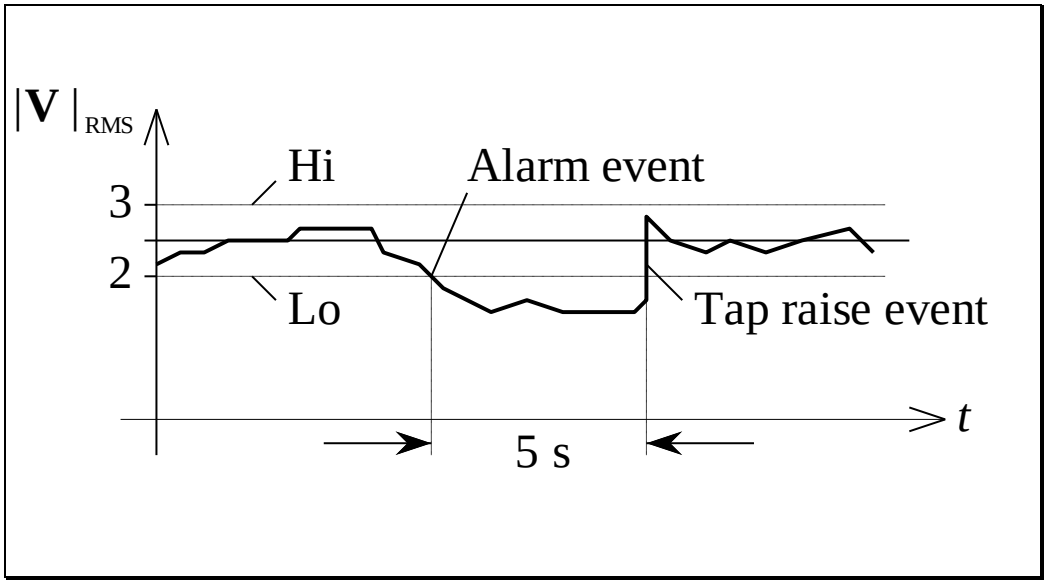

Figure P.2 – An example of the VRR in action

Alarm Output

On detection of the voltage deviating beyond the allowed limits, the VRR asserts the “Alarm” output, and starts timing. The “Alarm” output is cleared when the voltage falls back within the allowable range.

Timing Operation

The initial time delay between detection of an error in the monitored RMS voltage, to resultant tap change output, is selectable as either a definite time (5 seconds) or true inverse time response.

The initial time delay starts when the voltage deviation exceeds the upper or lower limit. If the deviation falls back to within set limits before the preset time delay is completed, the timer is reset. At the completion of the preset time delay the respective Raise / Lower output is set.

In the inverse time mode, the time delay is inversely proportional to the voltage deviation, down to a minimum of a one second delay, given by:

$$t = \frac{0.5\,\mathrm{V}}{|\mathbf{V}_{dev}|} \times 5\,\mathrm{s}$$

For example, for a detected error of 2 V (either high or low), the time delay is $1/4$ times the 5 s delay setting, or 1.25 seconds. The time delay should be updated every cycle based on the *rate* at which it should be timing, i.e. if the voltage deviation fluctuates during the time-out interval, then the rate of timing should also fluctuate.

Example of Inverse Timing

For example, with the 2 V deviation above, the timer should time out in 1.25 seconds, but if the deviation suddenly changes to 0.5 V at the 1 second mark (80% of the way to originally timing-out), then it should time out at a new rate of 5 seconds. Since it has already travelled 80% of the way to timing out, it only needs to time for 20% of the 5 second time, or 1 second. Thus, the total time out period is 2 seconds.

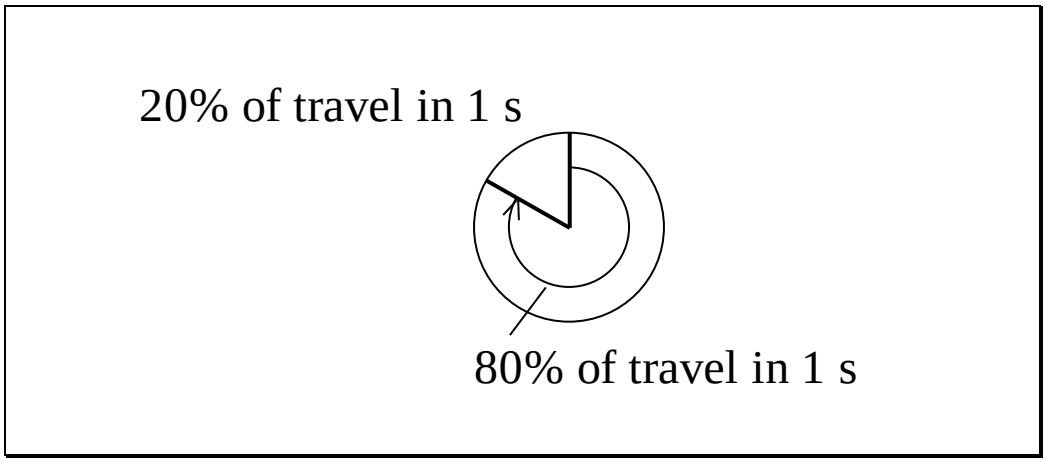

Figure P.3 – An example of inverse timing

Sample Timing

The μC needs to reliably take a sample value from the ADC every sample period. To do this, it needs to take into consideration any suspected interruptions such as PC communication and overhead such as SPI interfacing.

The time between samples is known as the sample period, and is usually denoted T_s. The value of T_s should be chosen so that there are at least 16 samples per cycle of the power system's sinusoidal voltage, which has a nominal frequency of 50 Hz, but which can vary between 47.5 Hz and 52.5 Hz.

Arithmetic Calculations

You are welcome to use floating-point calculations in your implementation, however they are not necessary with careful planning. The topic notes on fixed-point calculations should be studied carefully if you do not use floating-point calculations.

"Raise" and "Lower" Output Signals

The "raise" and "lower" output signals are to be 0 V when idle, and 5 V when active. The output signals are to be generated on Channel 1 and 2 (respectively) of the Tower outputs, utilising the analog board's DAC.

The number of times "Raise" and "Lower" have been set is to be stored by the VRR in non-volatile memory.

"Alarm" Output Signal

The "alarm" output signal is to be 0 V when idle, and 5 V when active. The output signal is to be generated on Channel 3 of the Tower outputs, utilising the analog board's DAC.

Frequency Tracking

Frequency tracking should be implemented on Phase A of the VRR only. If the voltage falls below 1.5 V then a default frequency of 50 Hz should be assumed.

Spectral Content

The magnitude spectrum of the voltage on Phase A should be calculated. The spectral resolution should be 50 Hz, up to a frequency of 350 Hz.

PC Interface

The interface to the VRR is via a PC running the Tower interface program under the Microsoft Windows® operating system. You are welcome to expand the Tower protocol for your own purposes.

Setting the VRR

The timing mode (definite or inverse) used by the VRR should be able to be set via the PC interface. The VRR should store the setting in non-volatile memory.

Interrogating the VRR

The PC should be able to interrogate the VRR for various quantities and settings, depending on the level of functionality.

Basic Protocol Extension for the VRR

Command Name	Command ID	Parameter 1	Parameter 2 (and 3 if needed)
Timing mode	0x10	0 = get 1 = definite (5 seconds) 2 = inverse	not used (0x00)
Number of Raises	0x11	0 = get 1 = reset to 0	not used (0x00)
Number of Lowers	0x12	0 = get 1 = reset to 0	not used (0x00)

Intermediate Protocol Extension for the VRR

Command Name	Command ID	Parameter 1	Parameter 2 (and 3 if needed)
Frequency	0x17	Hz×10 – low byte	Hz×10 – high byte
Voltage (RMS)	0x18	1 = phase A 2 = phase B 3 = phase C	Half-word (16-bits)

Advanced Protocol Extension for the VRR

Command Name	Command ID	Parameter 1	Parameter 2 (and 3 if needed)
Spectrum	0x19	0-7 = Harmonic number	Half-word (16-bits)

Software Development

There are two approaches to take for this simple embedded system – use a foreground / background approach, or use a real-time operating system (RTOS). The advantage of the foreground / background approach, for simple systems, is that it is easy to implement. For more complex systems, a real-time operating system simplifies the software design.

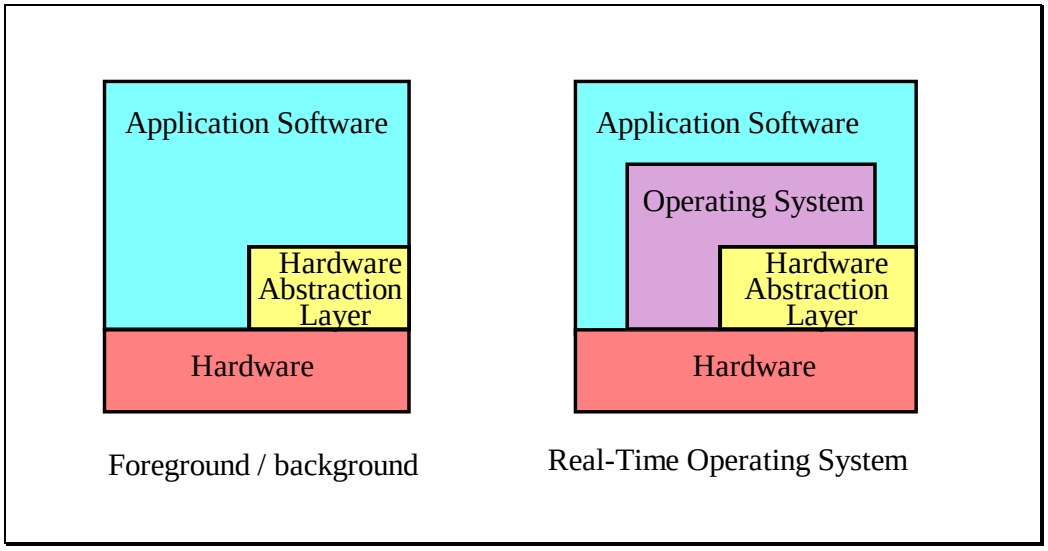

Figure P.4 – Two software approaches for an embedded system

For example, there are three inputs to the VRR. With a real-time operating system, only one piece of code (a program) needs to be written and tested – it is then used by three independent "threads". Management of time also becomes a lot easier with an RTOS. You will have the opportunity to use a simple RTOS.

RTOS

The RTOS is supplied as a library of object code with header files (no C source code is supplied – a typical scenario in industry). The documentation for the RTOS is supplied in a separate document.

Application

The VRR application code should exploit the RTOS capabilities. For example, threads could be created for: each of the input channels; communication with the PC interface program; frequency tracking, etc.

Strategy

The software should be developed in a modular fashion.

It is perfectly acceptable, and even advisable, to first get the VRR going using a foreground / background approach, perhaps with just one input channel. It is important in many projects to get parts of a system up and running quickly as a proof of concept of overall system design.

Also consider the fact that in the ultimate system, which uses an RTOS, all the shared code needs to be “re-entrant”, and communication via global variables should be kept to a minimum. Any communication between threads via global variables will need to be carefully examined to see if semaphores are needed. Timing and priority of threads will also be an issue.

Consult the Project Marking Scheme so you can manage your time and focus on areas that will maximise your marks based on your background, expertise and effort.