Project – Digital Overcurrent Relay

Overcurrent relay overview. Functional description. Software development.

Introduction

A digital overcurrent relay (DOR) is a power system protection device that initiates the tripping of a circuit breaker in the event of a fault. A DOR is better than an electromechanical relay because it is able to have a completely customisable *characteristic* (i.e. a $t \sim I$ tripping curve) 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 currents (labelled I_a, I_b and I_c) that require independent monitoring. The aim of this project is to implement a 3-phase DOR that is able to be set and interrogated remotely (via the PC and USB interface).

DOR Overview

The DOR 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 overcurrent situation. It needs to start operating instantly upon startup – thus a PC is unsuitable. It is a major task to develop hardware for the PC and write real-time drivers for the Windows® operating system. It is relatively easy to develop a stand-alone real-time embedded system.

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

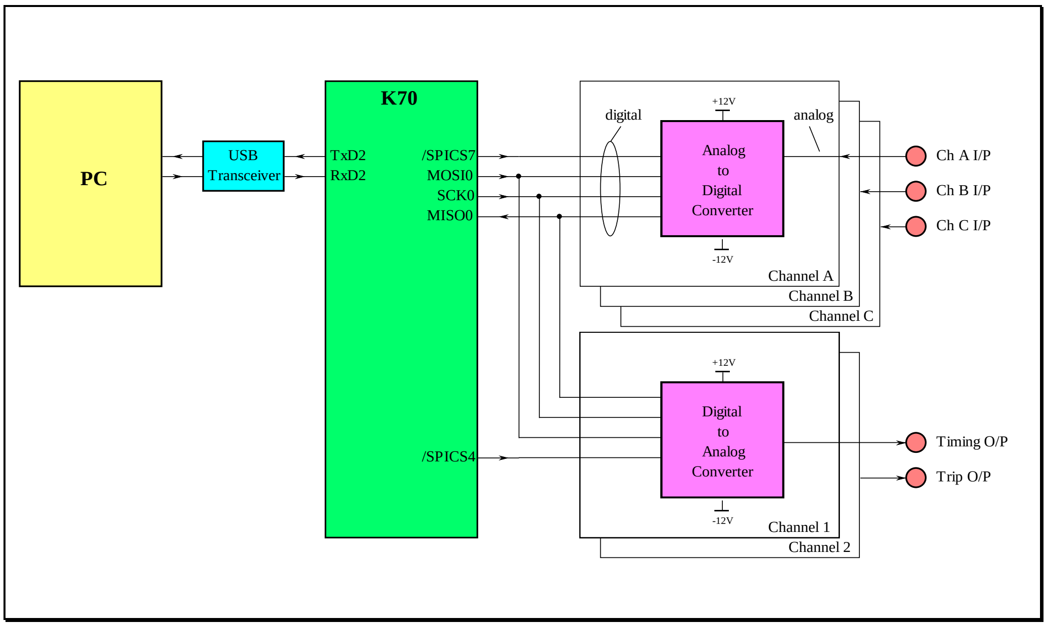


Figure P.1 – DOR Block Diagram

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

A USB interface is the primary means of communication with the μC. One of the six available UARTs on the μC is used for this purpose (UART2). The operation of the DOR 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 currents in the power system by additional signal conditioning circuitry which is not shown.

The μC is also responsible for generating the so-called “trip” signal which is used to open a circuit breaker, thus protecting the power system from an overcurrent condition. This is an output generated by the digital-to-analog converter (DAC) via the SPI. There is also a “timing” signal to signify that an overcurrent condition has been detected and the relay will shortly “trip”.

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.**

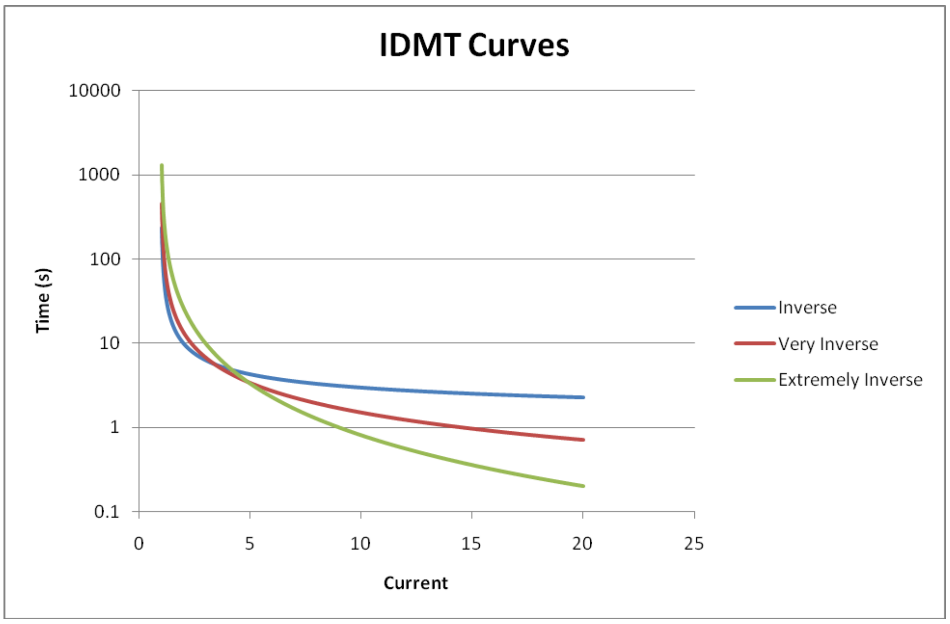
Functional Description

Functional Requirements

The DOR must meet the following functional requirements:

Specification	**Value**
Characteristics	Inverse, Very Inverse, Extremely Inverse
Frequency Range	47.5 Hz to 52.5 Hz
Current Range	0 to 20 times “rated” current
Accuracy	10% (current and time)
Measurement Method	True RMS
Sample Period	16 samples per cycle minimum

The relay *characteristic* is called an “inverse definite minimum time”, or IDMT curve. The IDMT $t \sim I$ curves are shown below, where the horizontal axis is the measured current and the vertical axis is the time until the relay “trips” the circuit breaker:



The characteristics are given by the following formula:

$$t = \frac{k}{I_{RMS}{}^{\alpha} - 1}, \quad I_{RMS} \geq 1.03 \qquad \text{and} \qquad t = \infty, \quad I_{RMS} < 1.03$$

where:

Characteristic	k	α
Inverse	0.14	0.02
Very Inverse	13.5	1
Extremely Inverse	80	2

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.

Some of the calculations for the required time delay until a "trip" event occurs are difficult to do in a fixed-point CPU. A simple strategy to circumvent difficult calculation is to lookup a value held in a hard-coded (compile-time determined) array and use interpolation for intermediate values.

Example of Inverse Timing

Suppose a particular value of I_{RMS} calls for a trip time of 1.25 seconds, based on the IDMT characteristic. After 1 second of timing (80% of the way to initially timing-out), the current suddenly changes so that the characteristic calls for a trip time of 5 seconds. Since it has already timed 80% of the way to tripping, it only needs to time for 20% of the 5 second time, or 1 second. Thus, the total time out period is 2 seconds.

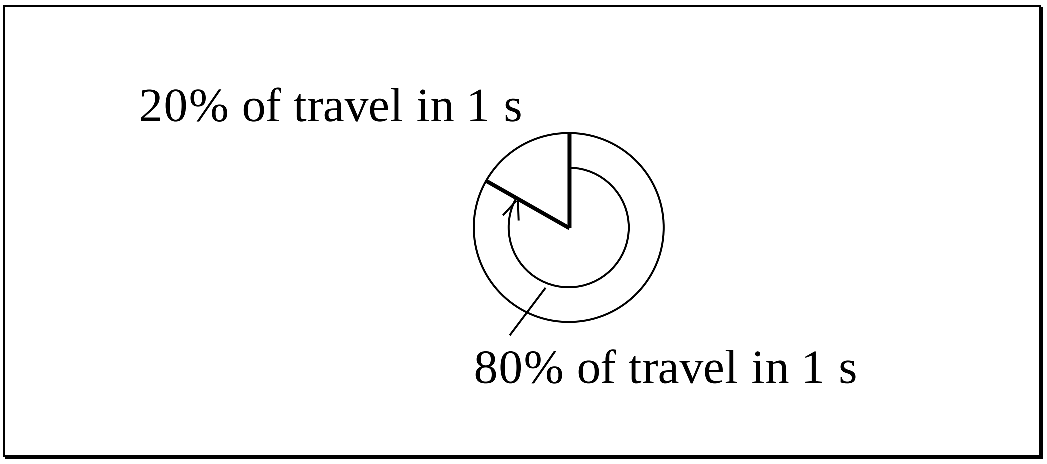


Figure P.2 – An example of inverse timing

Sample Timing

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 current, 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 may not be the fastest implementation (it depends on what operations you are doing). For this application (and many others) 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.

Note: Code with floating-point operations may not be optimal if speed of execution is part of the specification (and marking criteria).

Current to Voltage Conversion

We may assume that the signal conditioning circuitry is such that $I_{RMS} = 1$ is the "rated" current and corresponds to an input voltage of $V_{RMS} = 350\text{ mV}$.

"Sensitive" Fault Detection

The DOR is to implement a "sensitive" mode, in which it will only respond to the 50 Hz (or fundamental) component of the input waveform. i.e. all higher harmonics are filtered out before calculating the RMS value.

Phase Input Signals

The input signals (voltages) are to be applied to Channels 0 (phase A), 1 (phase B) and 2 (phase C) of the Tower inputs, which connect to the analog board's analog-to-digital converter.

Output Signals

The "Timing" output signal is to be 0 V when $I_{RMS} < 1.03$, and 5 V when $I_{RMS} > 1.03$ (for any phase A, B or C). The "Trip" output signal is to be 0 V when idle, and 5 V when active (operated by any phase A, B or C). The output signals are to be generated on Channel 1 and 2 (respectively) of the Tower outputs, utilising the analog board's digital-to-analog converter.

PC Interface

The interface to the DOR 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 DOR

The characteristic used by the DOR should be able to be set via the PC interface. The DOR should store the setting in non-volatile memory.

Interrogating the DOR

The PC should able to interrogate the DOR for the following: the characteristic in use (all phases have the same characteristic), the actual currents in all 3 phases at the instant of interrogation, the frequency of the current, the total number of times "tripped" (which is to be stored by the DOR in non-volatile memory) and the type of the last fault detected on the system (3-phase, 2-phase or 1-phase fault).

Packet Protocol Extension

Packets Transmitted from PC to DOR

The DOR should support the following received commands:

0x70	**DOR** Parameter 1: 0 = IDMT characteristic Parameter 2: 1 = get Parameter 3: 0
0x70	**DOR** Parameter 1: 0 = IDMT characteristic Parameter 2: 2 = set Parameter 3: 0 = Inverse 1 = Very Inverse 2 = Extremely Inverse
0x70	**DOR** Parameter 1: 1 = get currents Parameter 2: 0 Parameter 3: 0
0x70	**DOR** Parameter 1: 2 = get frequency Parameter 2: 0 Parameter 3: 0
0x70	**DOR** Parameter 1: 3 = get # of times tripped Parameter 2: 0 Parameter 3: 0
0x70	**DOR** Parameter 1: 4 = get fault type Parameter 2: 0 Parameter 3: 0

Packets Transmitted from DOR to PC

The DOR should support the following transmitted commands:

0x70	**DOR** Parameter 1: 0 = IDMT characteristic Parameter 2: 1 = setting Parameter 3: 0 = Inverse 1 = Very Inverse 2 = Extremely Inverse
0x70	**DOR** Parameter 1: 2 = frequency Parameter 2: LSB Parameter 3: MSB
0x70	**DOR** Parameter 1: 3 = # of times tripped Parameter 2: LSB Parameter 3: MSB
0x70	**DOR** Parameter 1: 4 = fault type Parameter 2: bit 0 = a, bit 1 = b, bit 2 = c Parameter 3: 0
0x71	**DOR Current** Parameter 1: 1 = phase number (0 = a, 1 = b, 2 = c) Parameter 2: LSB Parameter 3: MSB Note: The returned value is a 16-bit RMS value.

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.

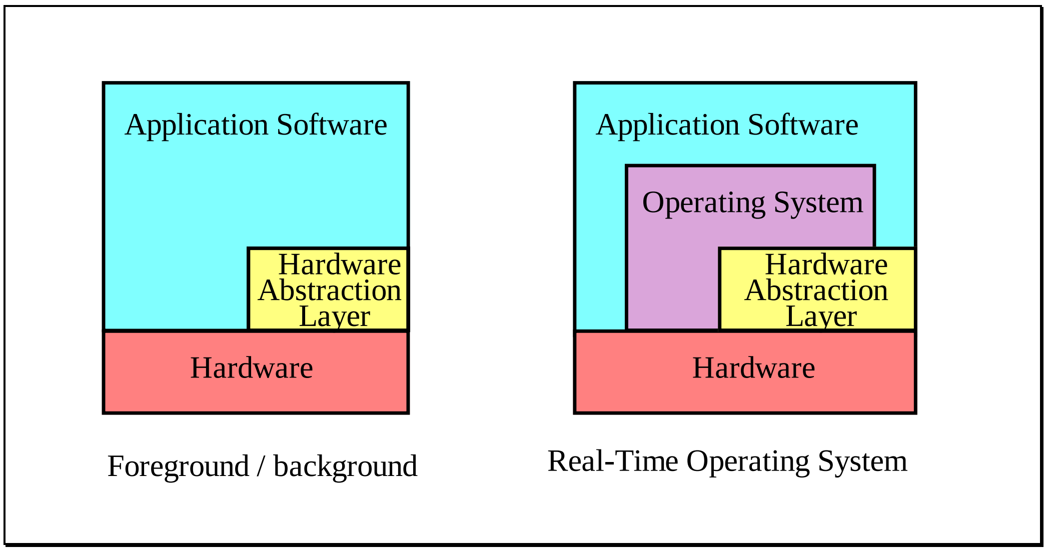


Figure P.3 – Two software approaches for an embedded system

For example, there are three inputs to the DOR. 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 DOR application code should exploit the RTOS capabilities – a thread should be created for each of the input channels, and a thread should be created for communication with the PC interface program, etc.

Strategy

The software should be developed in a modular fashion.

It is perfectly acceptable, and even advisable, to first get the DOR 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 of code that are important from an assessment perspective.