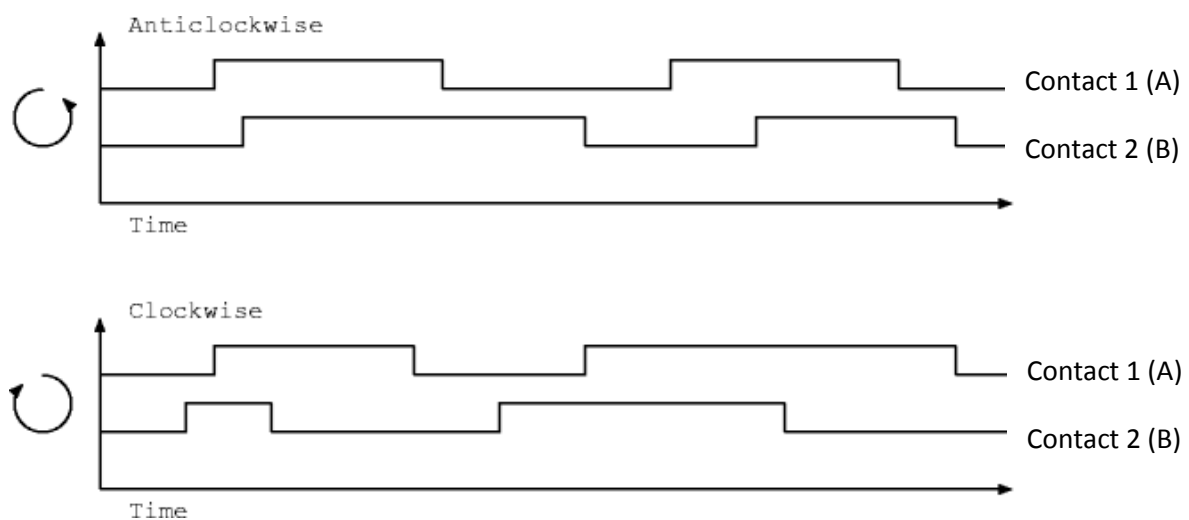


Workshop Two – Shaft Position Encoder

Introduction

Some industrial automation applications require control systems which know the rotational position of a shaft. Similar devices are also used for digital volume controls, etc, on domestic appliances. A shaft encoder is used to record the rotation typically in the form of a digital Gray code. In this workshop you need to build a decoder for such an encoder.

The shaft encoder you will be using produces a 2-bit Gray code. Such encoders are usually optical (e.g., those commonly used in a mouse), but in this case, we will be using a mechanical encoder. A disk is connected to the shaft which rotates with it. Electrical contacts are made with this disk to produce one of two Gray code sequences shown below depending upon the direction of rotation.



A shaft decoder module is required to convert this 2-bit Gray code (A,B) into a 4-bit position count. The 4-bit position should be incremented every time the input from the encoder changes owing to a clockwise rotation (e.g., from 00 to 01, or from 10 to 00). Similarly, the 4-bit position should be decremented every time the input from the encoder changes owing to an anticlockwise rotation (e.g., from 00 to 10, or from 01 to 00).

Components

- 1a prototyping box
- connecting wire
- rotary shaft encoder
- a PAL - GAL16V8
- 74HC193 - counter

Note: The recommended circuit for the shaft encoder is given in the lab booklet - remember to use the 4.7k Ω pull-down resistors to turn the switch open state into a firm logic value. Also note that the encoder provided steps through **all four** of the output combinations for each 'click' position.

Step 1 - shaft decoder to generate clockwise/anticlockwise signals

Design task 1: Draw a state diagram to decode this movement. Generate two independent

outputs, one that goes high for one clock cycle when the shaft rotates one output combination change clockwise, namely CW, and the other for one output combination change anticlockwise, namely AW. Assume 4 states, i.e., 00, 01, 11 and 10, and **make the next state equal to the current input** values A and B. For example, assume we are in state 00 and the shaft rotates one output combination anticlockwise giving current input values, A=1, B=0, the next state will be 10 and the outputs are AW=1 and CW=0. Now complete the state diagram for all 4 states and for all possible inputs per state (3 per state).

Design task 2: From the state diagram, create a state transition table. Note that this table will have columns for the Current State, the Next State (remember that is the same as the current inputs) and for the clockwise (CW) and anticlockwise (AW) outputs. Now determine expressions for the AW and CW outputs. Remember that it is possible to use don't care conditions to simplify the expressions for AW and CW. For example, one don't care condition for both AW and CW is for the 00 to 11 state transition that we know does not ever occur. We now wish to implement a synchronous (clocked from an independent free-running clock) finite state machine (FSM) inside a PAL.

The PAL has 8 possible output pins labelled P19 to P12 and 8 inputs labelled P2 to P9. Connect P2 to Contact 1 (A) and P3 to Contact 2 (B).

This exercise can be completed by designing a four-state FSM, where the D type flip-flops (FFs) having inputs labelled P17 and P16 hold this state. Your equations should look something like this:

```
P19 = p2
P18 = p3
P17 = p19
P16 = p18
P15 = <Some function of p16, p17, p18 and p19>
P14 = <Some function of p16, p17, p18 and p19>
```

This means that FF output P19 is a registered version of its input P2 (that is connected to A) and FF output P18 is a registered version of its input P3 (that is connected to B). The state of the FSM is held in the FFs having outputs P17 and P16 and these are registered versions of the FF outputs P19 and P18 respectively. Effectively, the Next State (that we have made equivalent to the current inputs) is FF outputs P19 and P18, and the Current State are FF outputs P17 and P16. The outputs labelled P15 and P14 are the Clockwise (CW) and Anticlockwise (AW) signals respectively and are therefore some function of the state (P17 and P16) and the registered inputs (P19 and P18).

NOTE: As indicated, we make the outputs from the shaft encoder (A and B) synchronous by using 2 FFs before these signals are applied to the 2 FFs that implement the FSM. To do this, make sure that the PAL compiler is in **registered mode**.

Implementation task 1: Now run your equations through the PAL compiler. Read the Appendix on PALs and the PAL compiler if you are unfamiliar with them. The compiler should be running on a stand-alone PC connected to the programmer.

Implementation task 2: Now connect your PAL to a rotary encoder and your two outputs to LEDs. Connect the clock to pin 1 and make sure that the output enable (pin 11) is connected to ground. To test your design start by setting the clock to 10Hz. Make sure that each LED flashes once as the rotary encoder is moved one output combination clockwise or anticlockwise.

Step 2 - count clockwise/anticlockwise pulses

Implementation task: Use the clockwise/anticlockwise signals as up/down signals for the 74HC193 counter. Connect the output of the counter to the 7-segment display. Make sure the unused inputs of the 74HC193 are tied off appropriately.

Optional Bonus Step

Task: How could you make use of two 74HC193 chips to provide an 8-bit up/down counter?

Assessment

Ticking criteria: Write up your design and answer the following questions. Demonstrate that your rotary position counter works. Once your work has met the **Common Ticking Criteria** (see Introduction), get your work ticked by an assessor. Remember that you need to hand in this assessed exercise as part of your portfolio of work (see the Head of Department's notice).

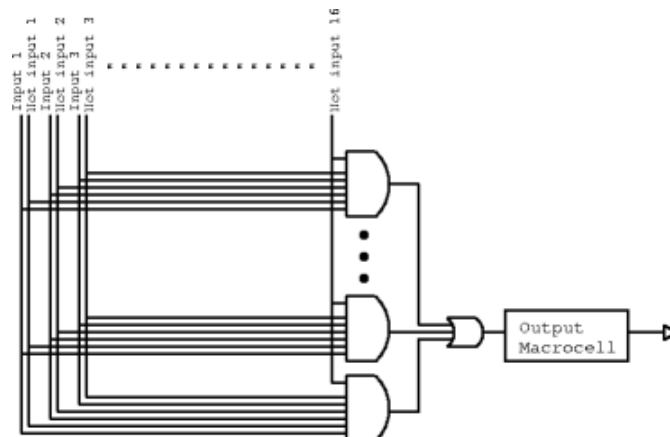
Questions

1. Why didn't we ask you to debounce the inputs from the rotary encoder?
2. How fast can the rotary encoder be rotated before your circuit fails to count correctly?

Appendix – PALs and the PAL Compiler

A PAL (programmable array logic) is a circuit which can be configured by the user to perform a logic function. It consists of an AND array followed by an OR array, the former being programmable. Inputs are fed into the AND array, which performs the desired AND functions and generates product terms. The various product terms are then fed into the OR array. In the OR array, the outputs of the various product terms are combined to produce the desired outputs.

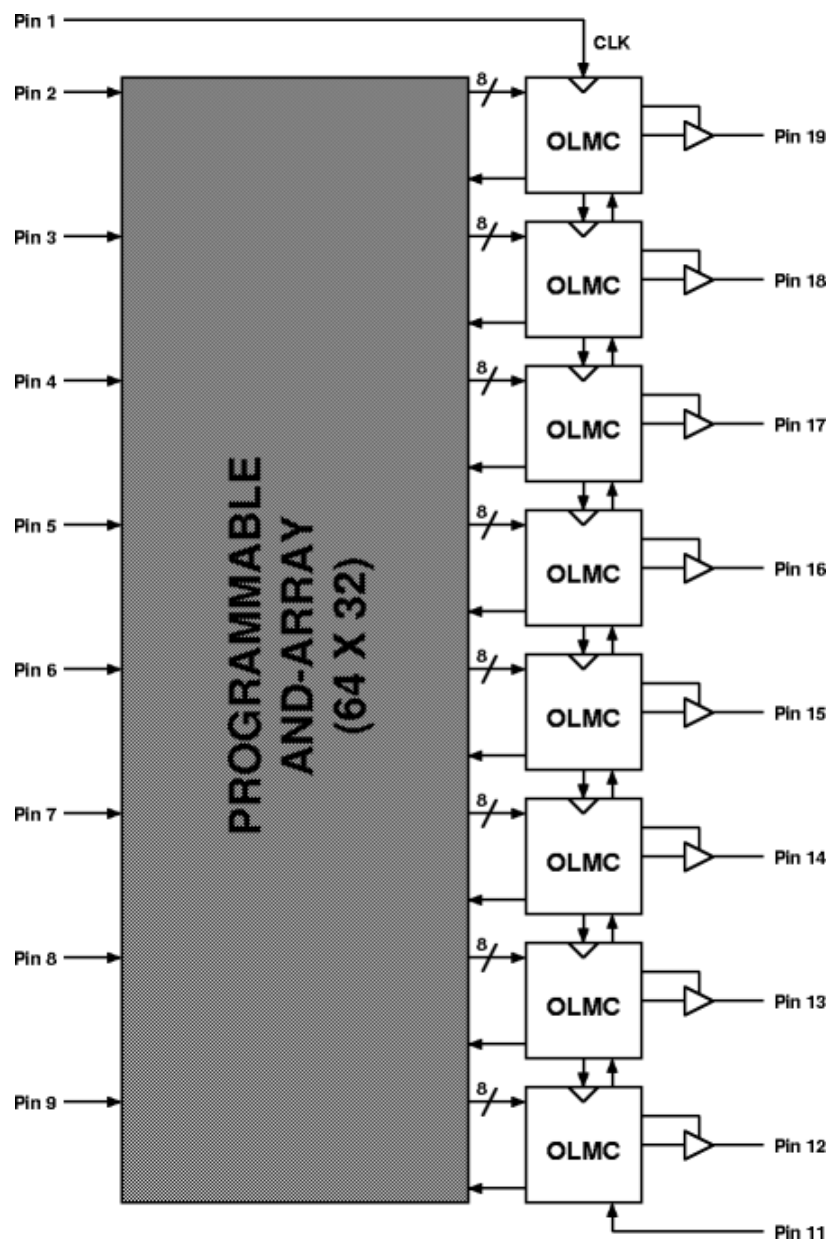
You will be using a small, relatively modern PAL called a GAL16V8. It has between 0 and 16 inputs and 8 to 0 outputs respectively, a common configuration is for up to 8 inputs and up to 8 outputs. Each output is driven by a single 8-input OR gate, which in turn is driven by eight 32-input AND gates. These 32 inputs can be programmed to be connected to the 16 possible inputs to the chip and their inverses. Note that when used to implement an FSM up to 8 of these inputs are actually outputs from the D-type FFs in each of the 8 output macrocells.



The output from the each OR gate is connect to an output macrocell. This can be programmed to latch (using a D-type FF) the output or feed the output signal back into AND array. For a more detailed description of this PAL, read the datasheet.

PAL Compiler

To program the PAL with the functions necessary to implement a state machine, you will need a PAL compiler. This will produce a JEDEC file which is needed by the programmer. You will be using a web-based compiler. You need to type in equations using variables p2 to p9 on the input side (representing the input pins) and p12 to p19 on the output side (representing output pins).



When using the PAL Compiler you should have the above diagram in mind. The compiler expects 8 equations, one to drive each output logic macrocell (OLMC). The OLMCs in turn drive pins 19 to 12. What the 16 inputs are depends on the configuration mode of the PAL. Pins 2 to 9 are always inputs, but in **registered** mode pins 12 to 19 make the other 8, in **complex** mode pins 1, 11 and 13 to 18 are used and in **simple** mode pins 1, 11 to 14 and 17 to 19. The mode can be selected using the check boxes.

The equations for each output (called P19 to P12, in that order) can consist of inputs, operators and brackets. The input must be referred to by the "P" followed by a number. Operators are &, |, ^ and ! for AND, OR, XOR and NOT respectively

Examples:

P19 = p2	A simple assignment latch
P19 = p2 ^ p3 P18 = p2 & p3	A half adder
P19 = !p19	A 1-bit counter

When you have typed in your equations, click the **compile** button.