Holtek’s new ultra low power consumption Flash MCU device series, the HT67Fxx series A/D with LCD type, and the HT69Fxx series LCD type, in utilising Holtek’s unique TinyPower™ technology, greatly reduce power requirements to meet the demands of today’s environmentally conscious products. Additionally, Holtek’s HT66Fxx A/D type series and its HT68Fxx I/O series further broaden the Flash range to provide an increased selection of flexible device solutions. The complete device range contains an extensive range of peripheral functions making them suitable for use in a diverse application area such as health care products, instrumentation products, household appliances, industrial control, consumer products, automotive peripheral products to name but a few.

- **HT68Fxx**: Internal Multi-function Timer Modules, IRC with four frequency selections. Multi-function I/O pins with re-mapping function. 4 LCD SCOM outputs for direct driving of LCD panels. Multiple Communication Interfaces.
- **HT66Fxx**: Includes all the HT68Fxx functions with an additional 12-bit ADC and internal reference voltage source.
- **HT69Fxx**: Uses Holtek’s TinyPower ultra low power technology. Internal R-type/C-type LCD driver.
- **HT67Fxx**: Includes all the HT69Fxx functions with an additional Internal High Speed 12-bit ADC with internal reference voltage source. Dual SPI or I²C interfaces. Internal Data EEPROM function etc.

**STD Flash MCU HT66/67/68/69Fxx Series**
- Wide Operating Voltage Range: 2.2V~5.5V
- Ultra Low Power
- Multiple Low-power Operating Modes
- Fast Wake-up Function
- Industrial Specifications: -40°C ~ +85°C
- Multiple External Communication Interfaces
- Internal Accurate High/Low Speed Oscillators
- High Noise Immunity and Excellent ESD protection
Control Shaft Encoders

Searching for an ideal control shaft encoder can be a time-consuming task. This article details some workarounds to help determine which encoder is appropriate for an application. An interesting debounce and decode circuit is also presented.

This article is about control encoders that are built from mechanical switches whose contacts alternate between conductive and non-conductive areas arranged in one or more circles. These devices are intended to be used as controls mounted on a panel with a knob on their shaft. They are also meant to be turned by human fingers. The devices are a subset of what are called “shaft” or “rotary” encoders, but these have generally fewer than 100 resolvable steps per rotation. Anything better requires an optical disk or another method with finer resolution (even some resistive potentiometers can be better than this under appropriate circumstances). Unless you have vintage wheels, your car radio almost certainly uses one of these types of low cost encoders for its volume control. The scroll wheel in your mouse is likely in this category as well—although the actual mouse position sensors usually use an optical disk in a mouse with a ball, or are purely optical in one without a ball.

My goal is to let you in on what I learned when I was doing my own search for an ideal control shaft encoder for a couple of my projects. I found that I was buying units that did not do the job. My frustrations included not knowing beforehand how the control would feel (just what do the torque specs mean!), not knowing what resolution was needed for a finger-rotated control shaft, and trying to gauge just how much debouncing was required.

As part of this article, I’ll also present a small debounce and decode circuit. I used a Microchip Technology PIC10F206 in an SOT-23 six-pin package (see Figure 1). All four I/O pins are used. Two inputs from the decoder and a clock pulse along with an up/down flag are used to drive an external counter. I wrote the code in Assembler and it served as a testbed. It can be adapted to any bigger PIC with little trouble since the PIC10 family’s instructions are a subset of larger PICs.

Whether they are described as “Gray code,” “Gray scale,” “incremental encoders,” “2-bit quadrature,” or even “phase difference signal” types, they all share this basic principle: at a constant rotation speed, they output two nominally square waveforms with what is intended to be a 90° overlap. It seems universal that the names of these signals are A and B, leaving the inspiring label “C” for the common. The relative polarity of the two waveforms might differ, but that does not matter. Just swap A and B if the opposite phase relation is desired. Also, it is not always clear which level, high or low, corresponds to a switch being closed or open. Sometimes the datasheets don’t show whether it’s assumed that the common is wired to the system’s positive voltage with pull-down resistors or wired to the ground with pull-ups. But again, it does not really matter since just swapping the signals will accomplish the direction reversal. So, either check out a real unit with a meter, or design your system to enable the easy swapping of the detected direction, either in hardware or software.

There are two “weasel words” in the last paragraph. Did you catch them? Yes, the waveforms are nominally square and the overlaps are intended to be 90°. Therein lies the real world. I’ll get into the real-world aspects later, but first the theory.

**CODE THEORY**

I think “2-bit quadrature” or “phase difference signal” are the most descriptive terms. Calling them Gray code devices is not wrong, but not really useful either. The 2-bit Gray code is effectively a degenerative case of general Gray codes. It is not necessary to know anything about Gray
codes here. All you need to know is that the overlapping codes that result from the waveforms in Figure 2 change only 1 bit at a time and that all four binary states of the 2 bits are represented. Moving from left to right in the figure, you see 00, 01, 11, and then 10. An important characteristic of this sequence is that if signal B leads signal A, the encoder is going in a certain direction. But if the lead/lag relationship is reversed, the encoder is rotating in the opposite direction. And, when you look at things in detail, for each and every case where a signal changes level, it can be determined which direction the shaft is rotating. This can be done by looking at 4 bits, two that are the state of the signals before the edge changed and two that are the state after the change. As an example, using Figure 1 again, if you have an upward edge on B when A is low (AB transition 00 to 01), the direction is the same (as if time flows from left to right in the figure). But if the upward edge on B happens when A is high, the opposite direction is indicated. I’ll show the code I used to implement this later.

**DETENT DETENT**

The biggest surprise I had when trying these units was how many different situations there are regarding the position of the detents with respect to the codes. Let’s define the electrical resolution as the angle determined by 360 × 4 pulses per revolution (PPR), and the mechanical resolution is 360 divided by the number of detents. I had originally thought that if a unit had detents, the detents would occur at every code position so that the mechanical and electrical resolutions would be the same. It turned out that I found four different detent schemes with three electrical-to-mechanical resolution ratios (see Figure 3). There are four lines above the code waveforms and each indicates with dots where detents can occur. The first line from the top shows just one detent per cycle. I show it when both A and B are high (contacts open), since that is what I found in all the units that had this resolution ratio. However, there is also the possibility of three other cases. The second and third lines show two cases with two detents per cycle, one with detents at A=B and the other with detents at A=B.

Now we have another “real-world” aspect to consider—the positioning of the detents. At first, it didn’t make sense that these were made with fewer detents than code positions. Why waste perfectly good codes? Besides, I found that it actually makes the decoding software more difficult. And, for a given mechanical resolution, the electrical resolution is two or four times finer, making the size of the contacts smaller by the same ratio. It turns out there is an advantage.

---

**Figure 1**—A simple testbed circuit. The firmware in the PIC10F206 needs to be changed depending on the type of encoder being tested. One thing to watch out for is that the processor cannot be programmed if either input A or B is held to the ground by an attached encoder. (The programming header is not shown.) If the encoder can be set to a position where both contacts are open, then that setup can be used. Otherwise, the encoder will have to be removed for programming.

**Figure 2**—A typical timing relationship of the A and B phases. Unlike most such diagrams, time can flow either way.

**Figure 3**—The detents can be built into the units with several relationships to the position codes. One, two, or four detents per cycle are all available.
Table 1—The encoders’ manufacturers, part numbers, and resolutions. The same numbers apply to Photo 1.

<table>
<thead>
<tr>
<th>Index</th>
<th>Manufacturer</th>
<th>Part number</th>
<th>Number of detents</th>
<th>PPR</th>
<th>Nominal size</th>
<th>Codes/detent</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Grayhill</td>
<td>25LB10-Q</td>
<td>36</td>
<td>9</td>
<td>1 inch</td>
<td>1</td>
<td>Acoustically noisy!</td>
</tr>
<tr>
<td>[2]</td>
<td>Panasonic</td>
<td>EYE-GA1F1724B</td>
<td>24</td>
<td>24</td>
<td>12 mm</td>
<td>4</td>
<td>Warning: similar units may contain barlings!</td>
</tr>
<tr>
<td>[3]</td>
<td>CUI</td>
<td>ACZ16NBR1E-15KQA1-24C</td>
<td>24</td>
<td>24</td>
<td>16 mm</td>
<td>4</td>
<td>All plastic housing</td>
</tr>
<tr>
<td>[4]</td>
<td>Bourns</td>
<td>3315C-001-006L</td>
<td>0</td>
<td>6</td>
<td>9 mm</td>
<td>0</td>
<td>Same as above, but with detents</td>
</tr>
<tr>
<td>[5]</td>
<td>CTS</td>
<td>28BT23R161A1</td>
<td>0</td>
<td>4</td>
<td>16 mm</td>
<td>0</td>
<td>PPR not verified</td>
</tr>
<tr>
<td>[6]</td>
<td>CTS</td>
<td>28BT23R161A2</td>
<td>16</td>
<td>4</td>
<td>16 mm</td>
<td>0</td>
<td>Cheapest (less than $1)</td>
</tr>
<tr>
<td>[7]</td>
<td>CTS</td>
<td>290VA5F201A1</td>
<td>0</td>
<td>20</td>
<td>9 mm</td>
<td>0</td>
<td>Appears similar to units 10 and 12</td>
</tr>
<tr>
<td>[8]</td>
<td>Mountain Switch</td>
<td>101-5437-EV</td>
<td>24</td>
<td>12</td>
<td>10 mm</td>
<td>2 (HH,LL)</td>
<td>Appears similar to units 9 and 12</td>
</tr>
<tr>
<td>[9]</td>
<td>ALPS</td>
<td>EC11B15202AA</td>
<td>30</td>
<td>15</td>
<td>11 mm</td>
<td>2 (HH,LL)</td>
<td>Cheap, appears similar to units 9 and 10</td>
</tr>
<tr>
<td>[10]</td>
<td>Pher</td>
<td>CI-11C9-V1Y22-HF4CF</td>
<td>30</td>
<td>15</td>
<td>11 mm</td>
<td>2 (HH,LL)</td>
<td>Cheap</td>
</tr>
<tr>
<td>[11]</td>
<td>Taiwan-Alpha</td>
<td>RE130F-40-20F-12P</td>
<td>12</td>
<td>12</td>
<td>12 mm</td>
<td>4</td>
<td>Cheap, form, fit, and function same as unit 3</td>
</tr>
<tr>
<td>[12]</td>
<td>BI</td>
<td>EN11-VNB1AO15</td>
<td>20</td>
<td>20</td>
<td>11 mm</td>
<td>4</td>
<td>Shortest</td>
</tr>
<tr>
<td>[13]</td>
<td>BI</td>
<td>EN12-VN20AF20</td>
<td>0</td>
<td>24</td>
<td>12 mm</td>
<td>0</td>
<td>Cheapest</td>
</tr>
<tr>
<td>[14]</td>
<td>BI</td>
<td>EN16-VZ2AF15</td>
<td>24</td>
<td>24</td>
<td>16 mm</td>
<td>4</td>
<td>Cheapest</td>
</tr>
</tbody>
</table>

More on this later.

The torque needed to turn the shaft is another aspect of detents. Actually, the torque itself is not the entire story. The knob used to drive the device’s shaft acts as a force multiplier. (Think of it as a differential force, with the thumb applying force in one direction and the forefinger in the other.)

According to Eastman Kodak’s book, Ergonomic Design for People at Work (Volume 1, 1983), the maximum torque that can be applied to a knob is in the order of 0.43 N-m (0.5” knob) to about 2 N-m for a 2” diameter knob. After 2”, the maximum torque falls off. It turns out that these numbers are all much larger than the largest torque spec I found in encoder datasheets. But there is still a significant range to the finger feel in these devices.

**The Lineup**

After encountering problems with three encoders from past projects, I went on a buying spree (the units cost anywhere from $0.50 to around $6) and bought 11 more. The first seven units were bought from distributor “D” and the last seven were obtained from distributor “M.” Photo 1 shows the devices and Table 1 lists the manufacturers, model numbers, resolutions (both PPR and number of detents), nominal sizes, and the code/detent relationships of all 14 units. In the rest of this article, I will refer to the units by their numbers, which you can see in the photo and the first column of the table.

All but two of the encoders have PCB through-hole terminals. The exceptions are units 5 and 6, which are in the same-lugged package. Of the PCB mount types, six are designed so that the encoder’s shaft is parallel to the PCB and six orient the shaft at right angles to the PCB. One can also obtain SMD versions of some of the smaller types.

As you can see in Photo 1, the devices differ greatly in size. Unit 4 is the smallest and unit 1 is the largest. There is no direct correlation between size and resolution, or between size and number of detents.

**Datasheet Despair**

The datasheet state of affairs is generally despicable. Several of these would rather say how high a voltage the contacts can handle [most designers won’t care that the unit that is going to be used to switch a logic level can handle 120 VAC] rather than inform the designer how the code pattern relates to the detents. I also found nonsensical units, cases where specifications were quoted without a clear indication of test conditions, and specifications stated in terms of entities that were not defined elsewhere in the datasheet. A situation I found at least once was signal timing diagrams that showed the detent positions as being in line with a signal edge. This was either a serious design flaw or a serious misunderstanding of how the units were designed on the part of the datasheet authors.

To describe the electrical resolution, datasheets can use any of the following: pulses per revolution, PPR, Number of detent [sic], and, in one case, pulses per revolution per track. Contact bounce is most often called just that, but it seems that chatter and sliding noise refer to the
LPC1200
Cortex-M0 – A simple choice

- Up to 128 KB Flash with 512 Byte page erase and 8 KB SRAM
- Industry’s first ROM-based divide library for Cortex™-M0, offering fast and deterministic execution for 32-bit division
- Unique configurable peripherals suited to energy efficient applications
- Supported by LPCXpresso tool chain for easy development and migration

www.nxp.com/cortex-m0
same problem: when a contact is sliding along a conductive section and it momentarily opens. Sometimes there is a specification for mechanical vibration.

There was an unusual number of odd or just wrong entries in the datasheets. I think electronics engineers are used to a fairly consistent standard with our ICs and discrete electronic component datasheets, but at this specific intersection of mechanical and electrical design, that consistency does not exist. I’ll just list some of the most unusual things I found. In the datasheet for unit 1, it clearly states that the insulation resistance between terminals is 1,000 MΩ. I suspect that instead of 1 Ω, 1,000 MΩ was meant, but 1 GΩ seems unusually high. In the datasheet for unit 2, I was confused by a word I had not seen before: some of the encoders covered by the document were built with “barlings.” The term was used consistently throughout, and it took Google “define: barling” [no definitions found] to convince me that this had to be a term for bearing! Another translation problem was found in unit 10’s document where the term “phrase difference” is used instead of “phase difference.”

The majority of the datasheets have contact bounce specifications in the range of 2 to 5 ms. Four of the datasheets do not have this very important specification [those that cover units 1, 3, 5, 6, and 8]! Often it is not clear at what shaft speed the debounce spec applies: a speed will be listed, but these are seemingly associated with either the lifetime specifications, or are a maximum speed, and not necessarily tied to the contact bounce spec. When one is using fingers to turn a shaft, a wide range of speeds can be obtained, and it would seem obvious that the slower you turn the shaft on this kind of switch, the longer the bounce time. So, if the bounce times are specified at either the maximum or lifetime test speeds, those specs will be next to useless! This might mean that to get absolutely reliable results from these units some sort of dynamic debounce logic or circuit would have to be used.

The only datasheets that supply a clear test condition for both the bounce time and the sliding noise are the last three, those from BI Technologies. In all of these three datasheets, the sliding noise is specified at 60 RPM and the bounce time is at 15 RPM.

A wide range of units is used to specify torque. I found all of the following: oz-in, mN·m, N·cm, gf·cm, and gf. [The last one has no length component and therefore is not a torque. This was on the datasheet for unit 8.]

One thing I never found any reference to in the datasheets is the acoustic noise level of the clicks when moving between detents. The first encoder I tried, unit 1 [Grayhill], was annoying in the application that I intended it for—lighting control in a quiet room—because it was just too loud!

**DECORATOR DECONSTRUCTION**

David Jones, host of the EEVblog, often says, “Don’t turn it on, take it apart.” I followed his advice and started doing just that.

The first unit did not need much deconstruction. Unit 1, the largest one, is made from a fairly thin PCB that’s heat-staked to the plastic body. Photo 2 is a view through the board from the back of the unit after a small label was removed. You can clearly see the encoder’s contact pattern and that there are nine FPR per track. The rotor must consist simply of three contacts connected together on the same radial line, with the contacts wiping near the centers of each track, connecting each outer one as a function of the angle to the center common one. It’s definitely a straightforward implementation. I was concerned that these radial connections to the pins would cause extra pulses at one particular location, but after thinking about it some more, I realized there wasn’t a problem. [I’ll leave this as a proverbial “exercise for the student.”] The contacts for units 5 and 6 can be seen through an opening, so I was able to conclude that they are of the same type of design.

Another observation: If you look at the angles covered by the contact area and the insulator between them, you will see that the contact area’s angle is the smaller of the two. In the first photo, where the two tracks are at quite different radii, a further difference is seen: the insulator’s angles seem to be about the same for both tracks, but the conductive areas are significantly different. Why? The sliding contacts themselves must necessarily cover an angle larger than zero, and this angular size needs to be subtracted from the conductive portions to approach the ideal performance.

I opened up unit 3. This proved to be quite a different beast [see Photo 3]. It uses a very different contact/track arrangement. As you can see in Photo 3a, the rotor is made up of three contacts in the same circle and equally.

---

**Photo 2**—The back of encoder 1. Note the nine contact areas in each of the tracks and the common track on the inside.
DISTANCE LEARNING COURSE

Programming Embedded PIC Microcontrollers

using Assembly, C and Flowcode

In this course you will learn how to program an embedded microcontroller. We will start with the absolute basics and we will go into a lot of detail. You cannot learn about software without understanding the hardware so we will also take a close look at the components and schematics. At the end of the course you will be able to design your own embedded applications and write the appropriate software for it.

Your course package:
• Courseware Ring Binder (747 pages)
• CD-ROM including software and example files
• Application Board
• Support at Elektor Forum
• Elektor Certificate

Price: $645.00 $575.00

Please note to be able to follow this course, E-blocks hardware is required which you may already have (in part). All relevant products are available individually but also as a set at a discounted price. Please check www.elektor.com/distancelearning for further information.

Further information and ordering at
www.elektor.com/distancelearning
spaced. The stator, on the other hand, is shown to be made up of three parts, each occupying a third of the circle (see Photo 3b). It's a clever transformation since it requires the same number of sliding contacts (three) and the same number of continuous metal sections for those contacts to slide on (three again), but there is no squeezing of tracks into smaller radii, which is required with the straightforward scheme.

The contact arrangement found in unit 3 turned out to be the most common scheme when I opened more and more units. Units 2, 8, 11, and 13 were all found to use the same scheme. It's likely units 9, 10, 12, and 14 also use the same scheme. That only leaves unit 4, which is totally sealed, and unit 7 in the undecided column. You can certainly understand the advantage of using one radius instead of three. Some of these units are so small that it is hard to imagine that two tracks (three counting the common) could fit in the radius available.

There is another type of design that appears in various sources I consulted. It is a scheme where one track is used but two sliding contacts are spaced with an effective offset of a quarter cycle. Of all 14 units, I found only one that uses this scheme: unit 7 (see Photo 4). It has contacts on opposite sides of the device's body but at the same radius. The contacts were in wells filled with silicone grease, so it was hard to immediately see that this was the scheme in use. Looking at the dimensions, the contacts are about 5.8 mm apart on a line through the center, so the required circumferential offset would be only a 0.23 mm (5.8 x 80 mm since this is a 20 PPR unit) difference in the lengths of the two contact arms, which seems rather hard to control. What seems to be the case is that the contact arms are the same length but both land on the code disk just a bit above the center line. I added a line through the center of the unit to show this (see Photo 4). Ideally, the offset
would be half of the calculated value, so something like 0.115 mm!

**CONTACT BOUNCE**

Now, let me justify the need for contact debounce. In an ideal situation, it would not be needed since, in theory, only one contact can change state at a time. But again, the real world intrudes. There are two situations that can cause both contacts to be changing at or near the same shaft angle. The first is when the bounce on one contact has not finished by the time the other contact has reached a transition. The second is when sliding noise (when a contact that is supposed to be closed opens briefly) occurs near either of the other contact’s transitions. Both of these conditions will be exacerbated by aging and will depend on shaft speed. In an optical encoder, there should be no need for debouncing, but that is not the case with these sliding contact types.

Useful information about contact bounce is available at the Ganssle Group’s website (www.ganssle.com/debouncing.htm). Bear in mind that most of the 16 switches dealt with there would have had some kind of spring-driven snap action that cannot be implemented in a position encoder. This can be seen by the fact that switch F, a slide switch, was found to be “very sensitive to the rate of activation.” That is what these encoders are: circular track slide switches. Both the R-C filtering approach and the use of the MCI4490 hex debouncer that Jack Ganssle covers are also suggested in several of the encoder datasheets.

My approach to debouncing was to take samples in a tight loop as quickly as possible but to require that N samples be the same before a change is declared. By changing the value of N, longer or shorter debounce times can be implemented (see Listing 1). The maximum number of instructions in the loop is 10 when the value read from the contacts matches the stored value, so it runs in 10 ms in the processor I used. A value for N (the constant **NEEDED** in the code) such as 32, which gives a debounce period of 320 ms, seemed to work fine. But wait a minute! Isn’t that about 10 times faster than the specs say? Welcome back to the real world. I had no means of measuring the shaft speed, but it was likely that since no knob was being used, the test speeds were on the low side. Like most design decisions, it is all about trade-offs. Here, the trade-offs are whether to ignore transitions that might happen at higher shaft speeds, but to be able to sense every single detent at low speeds, or whether to allow for the degrading effects of age, and it is a given that
the contacts will get worse with time.

**LOOK UP, LOOK WAY UP**

It only takes a simple 16-entry look-up table to affect the decoding of the combination of the two previous bits and the two new bits into a number, which tells the processor what to do. Also, half of the entries are zero because there are four combinations that are deemed incorrect (where both bits have changed) and four other combinations that represent no change at all. The pseudocode in **Listing 2** is a distillation of the Assembler code on the Circuit Cellar FTP site. All variable names are the same as in the Assembler code. The listing shows how the two least-significant bits (LSBs) of the look-up result are used to decide whether to increment or decrement the software counter internal_count or to do nothing. If the counter is modified, the next 2 bits are used to decide which encoder output had an edge change so that the two flags A_edge and B_edge can be set accordingly. If an edge happens on both A and B and the counter is such that its two LSBs are 0, then an output pulse is issued. The last sentence reflects the key difference between a one-code-per-detent and a four-code-per-detent encoder: if either of the tracks is seen to toggle without activity in the other, no change is reported. This is the advantage I alluded to earlier: fault tolerance. In a one-code-per-detent encoder, the external counter would be told to increment and decrement repeatedly under that condition. In a two-code-per-detent encoder, it turns out that at least one change on both phases is also required. A two-code-per-detent case would seem to be the ideal, but as Table 1 shows, these don’t seem to be as popular as the four-code-per-detent case.

To change the code to handle a two-code-per-detent encoder, only one change would be needed: require the LSB of the counter to be 0 to produce an output pulse instead of requiring the “10” condition. To use with a one-code-per-detent encoder, neither the internal counter

---

**Listing 1**—The simple debouncing routine. The constant NEEDED determines how many times through the loop the processor will run when the values do not change so that a new stable condition can be declared.

```asm
loop_start
    movf    GPIO,W            ;read in the port
    andlw   0x0C              ;just want two bits from it!
    movwf   entry             ;store these two bits into a variable
    subwf   old_sample,W      ;is the new value the same as the previous value?
    btfsc   STATUS,Z          ;if subwf result was zero then old and new are (still) the same
    goto    are_same
    ;since samples are different, need to re-start the debounce counter:
    movlw   NEEDED             ;initialize counter,
    movwf   needed_count       ;needed_count <- NEEDED
    movwf   entry,W            ;now put the new reading into old_reading
    movwf   old_sample         ;now the new is the old
    goto    loop_start
are_same
    ;the old and new match so decrement the counter if not at zero already:
    movf    needed_count,f    ;syntax is basically a test for zero
    btfsc   STATUS,Z           ;if needed_count is zero, go to at_end
    goto    at_end
    ;otherwise just decrement the counter:
    decf    needed_count,f
    goto    loop_start
at_end
```

---

**Listing 2**—Some pseudocode to show the logic of the look-up table. The variable names are taken from the actual code so that you can read the Assembler code easily.

```c
combo = (entry & 0x06) | ((combo >>2) & 0x03);
// i.e., combo is 2 new bits from encoder + the 2 old bits
CK_OUT = Low;   //Clear a possibly set clock output. Clear it here for a longer pulse width.
looked_up = table_start(combo);
if (looked_up != 0){
    if ((looked_up & 0x03) == 1) internal_count--;  
    if ((looked_up & 0x03) == 2) internal_count++;  
    if ((looked_up & 0x04) ) flag(B_edge) = true;
    else flag(A_edge) = true;
    if (flag(A_edge) & flag(B_edge) & ((internal_count&0x03)==0)){
        CK_OUT = high;
        flag(B_edge) = flag(A_edge) = false;
    }
}
```
nor the two edge flags are needed.

**ENCODERS IN HAND**

There are many important aspects to consider when using one of these encoders in a design. The datasheets are generally so poor that I can only conclude that in all cases you should get a sample or two before designing one of these in. For one-off, quick-and-dirty applications, I suggest having an assortment on hand. There are so many variations in mounting style, resolution, and size, I can't really suggest a list. But I hope Photo 1 and Table 1 will help. The other conclusion I offer is that device manufacturers generally need to improve how they prepare their datasheets.

Keith Brown, P.Eng., lives on Vancouver Island, BC. Operating as PLD Designs, which consults in the selection of PLDs and designs circuit boards for customers (with and without PLDs), he has published two earlier articles in Circuit Cellar. Go to www.island.net/~kdbrown for details. You may contact Keith by e-mail at kdbrown@island.net.

---

**PROJECT FILES**


**RESOURCES**

EEVBlog, www.eevblog.com


**SOURCES**

EC11B15202AA Encoder
ALPS Electronic Co. | www.alps.com

EN11-VNB1AQ15, EN12-VN20AF20, and EN16-V22AF15 Encoders
BI Technologies | www.bittechnologies.com

3315C-001-006L Encoder
Bourns, Inc. | www.bourns.com

288T232R161A1, 288T232R161A2, and 290VAA5F201A1 Encoders
CTS Corp. | www.ctscorp.com

ACZ16NBR1E-15KQA1-24C Encoder
CUI, Inc. | www.cui.com

25LB10-Q Encoder
Grayhill, Inc. | www.grayhill.com

PIC10F206 Microcontroller
Microchip Technology, Inc. | www.microchip.com

MC14490 Hex contact bounce eliminator
ON Semiconductor | www.onsemi.com

EVE-GA1F1724B Encoder
Panasonic Corp. | www.panasonic.com

CI-11C0-V1Y22-HF4CF Encoder
Piher International Corp. | www.piher-nacesa.com

RE130F-40-20F-12P Encoder
Taiwan Alpha Electronic Co. | www.taiwanalpha.com

---

**Leading Embedded Development Tools**

The complete development environment for ARM® Cortex™, 8051, and C166 microcontroller designs.

ARM 1-800-348-8051
www.arm.com

KEIL Tools by ARM
Reprogrammable UAV Autopilot System (Part 2)

Testing and Results

In the first part of this series, you were introduced to the SLUGS reprogrammable UAV autopilot system. This article details the system's software and flight test results.

In the first part of this two-part series, we introduced the hardware used in the Santa Cruz Low-cost UAV GNC System (SLUGS), an open-source unmanned aerial vehicle (UAV) autopilot primarily focused on supporting guidance, navigation, and control (GNC) research. In this article, we’ll focus on the software architecture and algorithms. We’ll also present results from temperature compensation, sensor calibration, and flight tests.

SOFTWARE ARCHITECTURE

The SLUGS has two Microchip Technology dsPIC33 digital signal controllers (DSCs) that serve as its main processing units, running at 40 MIPS. These are interconnected via a SPI at 10 MHz and are called the “sensor” DSC and “control” DSC, respectively. The sensor DSC is in charge of reading the sensor data and converting it into a high-quality estimate of the aircraft position and attitude [3-D orientation]. The control DSC interacts with the user through the ground station, generating signals to the UAV’s control surfaces to stabilize the aircraft and fly it along the user-designated mission (at its highest level), traversing waypoints at a given altitude and airspeed.

One of the key advancements in the SLUGS autopilot is that the DSCs are programmed directly from MATLAB Simulink using a block and signal flow metaphor for visual programming. This offers several advantages. Most researchers in the GNC field are already familiar with MATLAB and use it to prototype and simulate new algorithms. Complicated algorithms and changes are easy to implement, rapidly increasing the prototype-simulate-test cycle. Lastly, the software can be easily retargeted for better processors without having to rework the low-level, back-end code. While this environment sacrifices some efficiency, any performance issues are made insignificant by the speed with which changes can be made and tested.

The sensor DSC’s low-level software architecture (see Figure 1) is a combination of blocks provided by Lubin Kerhuel’s dsPIC Simulink Embedded Target (ET) (shown in

![Figure 1—Low-level software architecture for the sensor DSC](image-url)