CommonBusEncoders

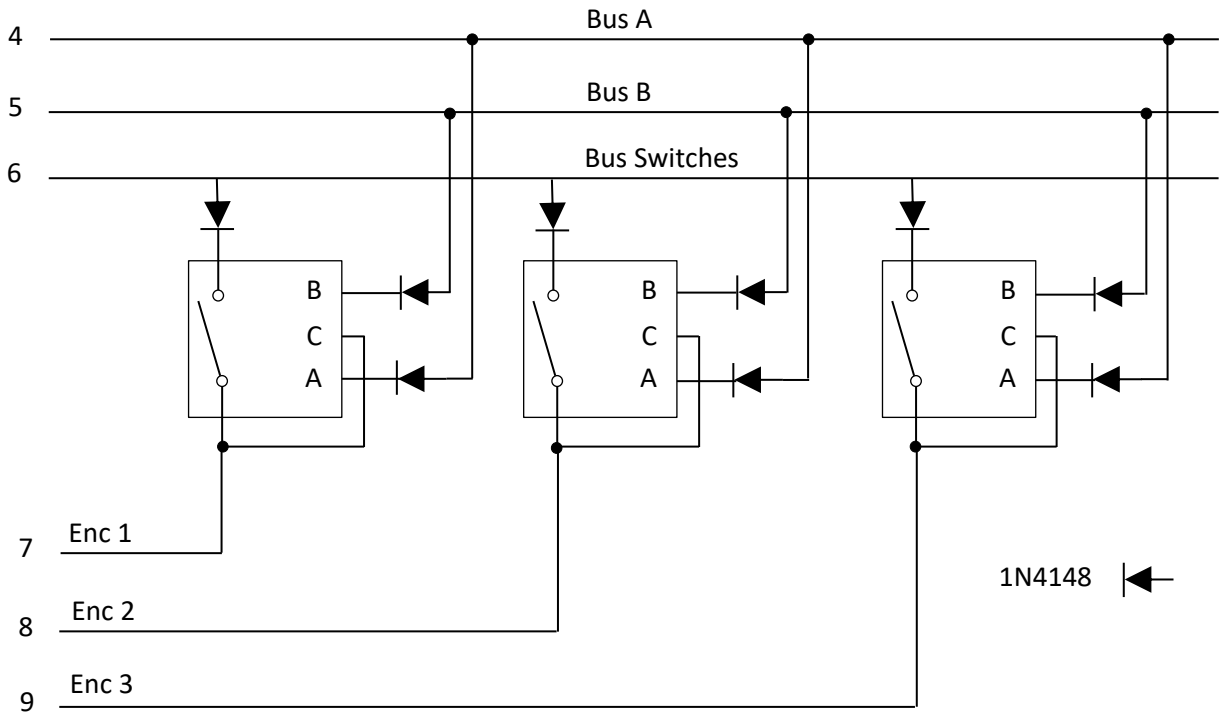
TUTORIAL

WHAT IS CommonBusEncoders?

The CommonBusEncoders Library allows you to create an encoder bank. The first encoder requires four pins to connect to Arduino. Each of the other encoders requires a single extra pin.

HOW DO WE WIRE THE ENCODERS?

Here is an example with 3 encoders (We can have as many encoders as the Arduino has digital pins):



HOW DOES IT WORK?

When we describe an encoder to the CommonBusEncoders Library, we give it a specific code for the encoder itself, and another one for its switch.
When this encoder is rotated, the CommonBusEncoders Library will return its code if it was turned clockwise, and its code + 1 if it was turned counter clockwise.

The switch can behave in two ways. First, it can act as a normal switch. In this case, the CommonBusEncoders Library will return its code. Second, it can act as a mode switch. This allows using the encoder to change multiple variables. Pressing the switch will place the encoder on the next mode. When the encoder is on its last mode, pressing the switch will revert the encoder to its first mode. With four modes, the presses on the switch will get the encoder into the following modes:
MODE 0 -> MODE 1 -> MODE 2 -> MODE 3 -> MODE 0 -> MODE 1 etc.

As an example with an encoder that has four modes, and a code of 100, the values returned by the CommonBusEncoders Library will be:

MODE 0		MODE 1		MODE 2		MODE 3	
CW	CCW	CW	CCW	CW	CCW	CW	CCW
100	101	102	103	104	105	106	107

WILL IT WORK WITH ANY ENCODER?

The encoders that are handled by the CommonBusEncoders Library are of the "**Quadrature encoders**" type. The Library is to be used for encoders that will be turned by a human hand. Motor driven encoders turn too fast for this Library. For a good tutorial about quadrature encoders, please read: http://playground.arduino.cc/Main/RotaryEncoders.

It does not matter if the encoders are mechanical or optical. If mechanical encoders are used, a debouncing layer is available and can be set to 32 different levels. Optical encoders do not need debouncing. If we use the two types in the same setup, we should use the debouncing layer. As default, the debouncing layer is set to 16, but it can be changed in the sketch. We can disable the debouncing layer by setting it to level 1.

Furthermore, this Library is designed for encoders that have detents (clicks). Some encoders have 4 steps per click, and others have 2 steps per click. This Library will be able to handle either one or a mix of them. The Library will return a code to the sketch only when the encoder clicks. This design has been made because it is used to change **discrete values** (integers, letters of the alphabet, angle in degrees…).

Those choices allow the Library to operate without the need for interrupts (hardware or software). A test was made with 17 cheap encoders (CAN$8.55 for a pack of 10) without any noticeable lags or misses (Unless we turn the encoders really fast). Remember that if you are to use interrupts, the debouncing has to be made in hardware. Jack G. Ganssle from the Ganssle Group is clear about this. Interrupts will react to a bounce, and it will be too late to do anything about it once we are in the interrupt service routine (ISR). See this guide: http://www.eng.utah.edu/~cs5780/debouncing.pdf.

USING THE COMMON BUS ENCODERS LIBRARY

The library is invoked with:

```
#include <CommonBusEncoders.h>
```

The first thing we have to tell the CommonBusEncoders Library is where the three busses are connected to the Arduino. We also need to specify how many encoders that we have in our setup. They have to be supplied in this exact order: "Bus A", "Bus B", "Bus Switches", "Number of encoders". For the above example, supposing that we want to name our setup **encoders**, we will write:

```
CommonBusEncoders encoders(4, 5, 6, 3);
```

This code has to appear before the setup() section of the sketch.

Next, for each encoder, we will supply:

- The number of the encoder (starting from 1);
- The type of the encoder :
  - 2 if two steps per click;
  - 4 if four steps per click;
- The pin of the encoder (The one that is connected to the common pin);
- The number of modes for this encoder;
- The code that will be returned for this encoder;
- The code that will be returned for this encoder's switch:
  - 0 if the encoder is multi-mode;
  - any code if it is a single mode encoder;

Those declarations have to reside in the setup() section of the sketch.

Example :

- Encoder 1 is of the 4 steps per click type. Its common pin is tied to Arduino's pin 7. It will control two variables (2 modes). The codes we want to receive are:
  Mode 0, CW: 100;
  Mode 0, CCW: 101;
  Mode 1, CW: 102;
  Mode 1, CCW: 103;
  The switch is used to change modes.

- Encoder 2 is of the 4 steps per click type. Its common pin is tied to Arduino's pin 8. It will control four variables (4 modes). The codes we want to receive are:
  Mode 0, CW: 200;
  Mode 0, CCW: 201;
  Mode 1, CW: 202;
  Mode 1, CCW: 203;
  Mode 2, CW: 204;
  Mode 2, CCW: 205;
  Mode 3, CW: 206;
  Mode 4, CCW: 207;
  The switch is used to change modes.

- Encoder 3 is of the 4 steps per click type. Its common pin is tied to Arduino's pin 9. It will control one variable (1 mode). The codes we want to receive are:
  Mode 0, CW: 300;
  Mode 0, CCW: 301;
  The switch will return code 399;

Add these 3 lines in the setup() section of the sketch:

```
Void setup() {
  encoders.addEncoder(1, 4, 7, 2, 100,   0);
  encoders.addEncoder(2, 4, 8, 4, 200,   0);
  encoders.addEncoder(3, 4, 9, 1, 300, 399);
}
```

To read the encoders, in the loop() section of the sketch, place this line:

```
Void loop() {
  int code = encoders.readAll();
  if (code > 0) { //Do something with the code; }
}
```

WHAT ARE THE METHODS THAT THE MENU LIBRARY PROVIDES?

MANDATORY METHOD

CommonBusEncoders::CommonBusEncoders(int PinA, int PinB, int PinS, int count) (MANDATORY)

```
CommonBusEncoders encoders(4, 5, 6, 3);
```

The first method is called a constructor, which creates the encoders bank. It has to be placed **before** the setup() section of the sketch.
PinA is the pin to which is connected the Bus A line**;**
PinB is the pin to which is connected the Bus B line**;**
PinS is the pin to which is connected the Bus Switches line**;**
count is the number of encoders you have in your setup.

You can have as many encoder banks as you wish. Just give them different names.

METHOD TO ADD AN ENCODER TO THE BANK

void CommonBusEncoders::addEncoder(int id, int type, int pin, int modes, int codeE, int codeS)

```
encoders.addEncoder(1, 4, 7, 2, 100,   0);
encoders.addEncoder(2, 4, 8, 1, 200, 299);
```

You will use this method to add an encoder to the encoder bank.
id is the number of the encoder, starting with 1;
type is the number of steps per detent of the encoder (2 or 4);
pin is the Arduino pin tied to the encoder's common pin;
modes is the number of modes (variables) controlled by the encoder;
codeE is the code to be returned when the encoder is turned;
codeS is 0 if the encoder has multiple modes or the code to be returned if the switch is pressed.

METHOD TO READ THE ENCODERS

void CommonBusEncoders::readAll()

```
code = encoders.readAll();
```

You will use this method to get the code that is attached to the encoder that is being turned or the code attached to its switch if the encoder is not used as multi-mode. If there is no activity on any encoder, you will get zero (0).

OTHER USEFULL METHODS

void CommonBusEncoders::setDebounce(int reads)

```
encoders.setDebounce(16);
```

You will use this method to specify the debounce limit.
reads is the number of times the pin is read to confirm a solid HIGH or a solid LOW (1 to 32)

The algorithm reads the pin a specified number (N = 1..32) of times repeatedly. When the number of reads is done, the result will be returned to the upper layer only if all the reads are HIGH or all the reads are LOW. If some reads differ from the others, it means that the encoder is rotating and it's A or B pin is bouncing. The algorithm will go around for another N reads, up until the read is solid, at which point the value will be returned to the upper layer.

Use this method to adapt the library to the bouncing of your encoders. The bounciest encoder will determine the limit you will use.

If the method is not called, a limit of 16 reads is used.

void CommonBusEncoders::resetChronoAfter(int someTime)

```
encoders.resetChronoAfter(1000);
```

You will use this method to set the time an encoder will keep the focus of the Library after the last time it was turned or that its switch has been pressed.
someTime is in milliseconds.

In order to keep up with the encoders, when an encoder is being turned, the Library will focus on it, not reading the other encoders. As long as the encoder is turned, Chrono is set back to zero. When there is no activity on that encoder, chrono will eventualy reach **someTime**, and the encoder loses focus. All the encoders are now scanned for possible activation. When a multi-mode encoder loses focus, it is put back to MODE 0.

Change the value to suit your use of the encoders. The default value is 500 milliseconds.

bool CommonBusEncoders::focussed()

```
if (!encoders.focussed()) { //Check other switches or do something else; }
```

You will use this method to find out if an encoder has the focus of the library. If some work can be delayed until the encoder stops moving (plus **someTime**), you can help the library continue reading the encoder without missing clicks.

BY THE WAY…

In order to keep memory requirements as low as possible, only the space required for the number of encoders for your project is used. When the constructor is called, the library uses the argument "count" and reserves 14 bytes for each encoder plus 14 bytes for its own use. That memory is not reported by Arduino's IDE. It is requested after compile time.

Say Arduino's IDE reports 430 bytes used by the variables and that your project uses 6 encoders, then the real memory used by the variables will be: 430 + (6 x 14) + 14 = 528.

EXAMPLE CODE (Putting it all together)

```
#include <CommonBusEncoders.h>

CommonBusEncoders encoders(3, 2, 10, 3);

void setup() {
  encoders.setDebounce(8);
  encoders.resetChronoAfter(2000);
  encoders.addEncoder(1, 4, 8, 2, 100,   0);
  encoders.addEncoder(2, 4, 6, 4, 200,   0);
  encoders.addEncoder(3, 4, 4, 1, 300, 399);
  Serial.begin(9600);
}

void loop() {
  int code = encoders.readAll();
  if (code != 0) Serial.println(code);
}
```

I sincerely hope that this CommonBusEncoders Library will help you in your projects.
Jacques Bellavance