KY040

1.0.1

Generated by Doxygen 1.10.0

Chapter 1

KY040

An Arduino library for KY-040 rotary encoders. The library has debouncing and works in polling mode, with pin change interrupts or normal interrupts. In polling or pin change interrupt mode you can attach more then one rotary encoder to your Arduino Uno/Nano.

Examples how to use the library

- examples/pollingNoInterrupts/pollingNoInterrupts.ino
- examples/pinChangeInterrupt/pinChangeInterrupt.ino
- examples/pinChangeInterruptPowerSave/pinChangeInterruptPowerSave.ino
- examples/pinChangeInterruptDualEncoders/pinChangeInterruptDualEncoders.ino
- examples/withInterrupt/withInterrupt.ino

1.1 License and copyright

1.2 Appendix

1.2.1 Background

KY040 is library for KY-040 rotary encoders. There are a lot of libraries existing for KY-040, but I found no library (at least 12/2023) which

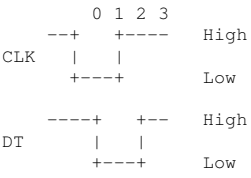
1. works without the need of interrupt enabled pins for the CLK (aka. A) and DT (aka. B)
2. and has a stable debouncing using a signal state table (without debouncing the KY-040 rotary encoder is a mess)

So I wrote my own library KY040, which was designed to work on an Arduino Uno/Nano or ATmega328 and could work on other Arduino compatible MCUs too.

The KY040 library can:

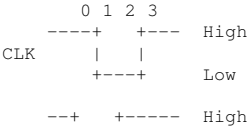
- be used without interrupts in polling mode
- be used with pin change interrupts
- control more than one rotary encoders in polling or pin change interrupt mode on an Arduino Uno/Nano
- use any common pin digital pins for CLK and DT in polling or pin change interrupt mode
- be used with normal *attachInterrupt* interrupts (in this case could have to use Pins 2 and 3 on your Arduino Uno/Nano)
- be used with SLEEP_MODE_PWR_SAVE/SLEEP_MODE_PWR_DOWN sleep mode in combination with pin change interrupts
- debounce the rotary encoder by filtering out invalid signal sequences

1.2.2 Valid clockwise sequence



Step	Signal level for CLK/DT
0	Low/High
1	Low/Low
2	High/Low
3	High/High

1.2.3 Valid counter-clockwise sequence



```
DT     |    |
       +---+        Low
```

Step	Signal level for CLK/DT
0	High/Low
1	Low/Low
2	Low/High
3	High/High

1.2.4 KY-040 Hardware

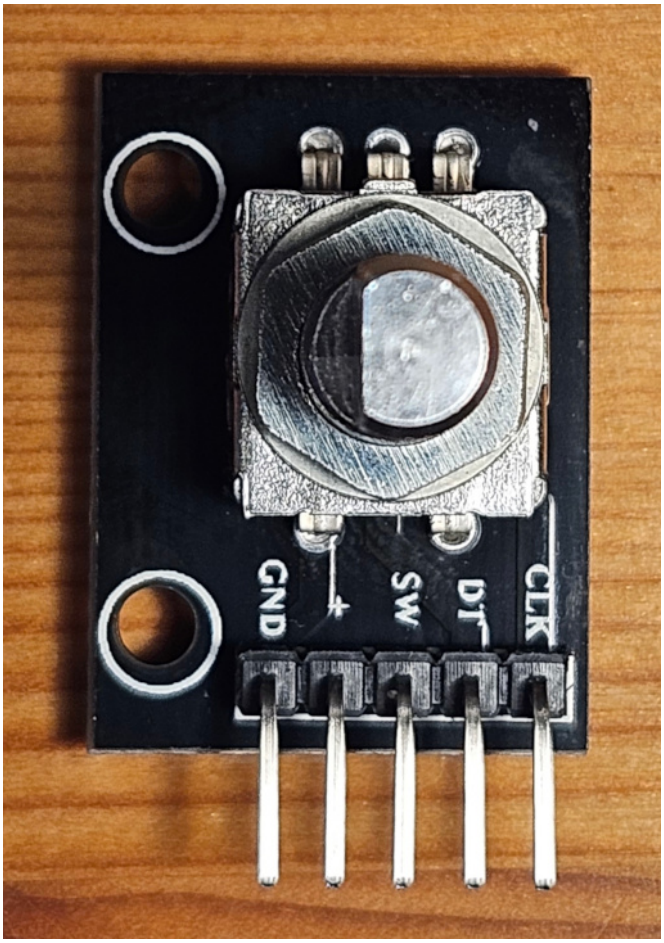


Figure 1.1 Frontside

Pins	Comment
GND	Ground
+	Vcc
SW	Switch button, not covered by this library. You can use *digitalRead* statements to check SW. Pin is pulled up to Vcc via the 10k pullup resistor R3
DT	aka. B, Pin is pulled up to Vcc via the 10k pullup resistor R2
CLK	aka. A, Pin is pulled up to Vcc via the 10k pullup resistor R1

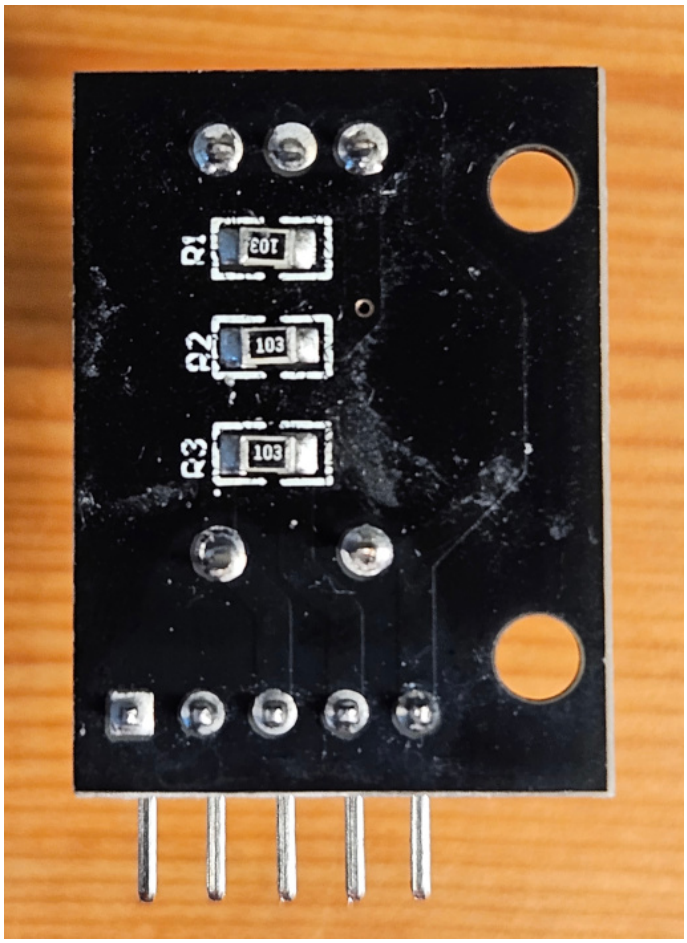


Figure 1.2 Backside

The three 10k resistors R1,R2 and R3 pulls up the SW, CLK and DT pins up to Vcc.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 KY040 Class Reference

`#include <KY040.h>`

Public Types

- enum directions { IDLE , ACTIVE , CLOCKWISE , COUNTERCLOCKWISE }

Public Member Functions

- KY040 (byte clk_pin, byte dt_pin)
  Constructor of a the KY040 rotary encoder.
- byte checkRotation ()
  Returns current rotation state from stored pin state.
- byte getAndResetLastRotation ()
  Get and reset last finished rotation step (Do not use inside ISR)
- byte getRotation ()
  Read and stores current pin state for CLK and DT and returns the current rotation state.
- byte getState ()
  Get stored pin states for CLK and DT (Left bit is for CLK, right bit is for DT). Should be called from ISR, when needed.
- bool readyForSleep ()
  Checks, if it save to go to sleep.
- void setState (byte state)
  Stores pin states for CLK and DT (Left bit is for CLK, right bit is for DT). Should be called from ISR, when needed.

4.1.1 Detailed Description

Class for a KY-040 rotary encoder

Definition at line 62 of file KY040.h.

4.1.2 Member Enumeration Documentation

4.1.2.1 directions

`enum KY040::directions`

Rotation states

Enumerator

IDLE	Rotary encoder is idle
ACTIVE	Rotary encoder is rotating, but the CLK/DT sequence has not finished
CLOCKWISE	CLK/DT sequence for one step clockwise rotation has finished
COUNTERCLOCKWISE	CLK/DT sequence for one step counter-clockwise rotation has finished

Definition at line 65 of file KY040.h.

```
00066     {
00067       IDLE,
00068       ACTIVE,
00069       CLOCKWISE,
00070       COUNTERCLOCKWISE
00071     };
```

4.1.3 Constructor & Destructor Documentation

4.1.3.1 KY040()

```
KY040::KY040 (
            byte clk_pin,
            byte dt_pin )  [inline]
```

Constructor of a the KY040 rotary encoder.

Parameters

in	*clk_pin*	Digital input pin connected to CLK aka. A
in	*dt_pin*	Digital input pin connected to DT aka. B

Definition at line 79 of file KY040.h.

```
00080     {
00081       m_clk_pin = clk_pin; // aka. A
00082       m_dt_pin = dt_pin; // aka. B
00083       v_state = 255;
00084       v_lastResult = IDLE;
00085       v_lastSequenceStartMillis = millis();
00086       v_sequenceStep = 0;
00087       v_direction = IDLE;
00088       v_oldState = INITSTEP;
00089     }
```

4.1.4 Member Function Documentation

4.1.4.1 checkRotation()

```
byte KY040::checkRotation ( )  [inline]
```

Returns current rotation state from stored pin state.

If you do not use interrupts, you have to start setState() and checkRotation() or a function using these (for example getRotation()) very frequently in your loop to prevent missing signals

Return values

KY040::CLOCKWISE	CLK/DT sequence for one step clockwise rotation has finished
KY040::COUNTERCLOCKWISE	CLK/DT sequence for one step counter-clockwise rotation has finished
KY040::IDLE	Rotary encoder is idle
KY040::ACTIVE	Rotary encoder is rotating, but the CLK/DT sequence has not finished

Definition at line 102 of file KY040.h.

```
00103     {
00104       byte result = IDLE;
00105
00106       if (v_state != v_oldState) { // State changed?
00107         if (v_sequenceStep == 0) { // Check for begin of rotation
00108           if (v_state == c_signalSequenceCW[0]) { // Begin of CW
00109             v_direction=CLOCKWISE;
00110             v_sequenceStep = 1;
00111             v_lastSequenceStartMillis = millis();
00112           }
00113           if (v_state == c_signalSequenceCCW[0]) { // Begin of CCW
00114             v_direction=COUNTERCLOCKWISE;
00115             v_sequenceStep = 1;
00116             v_lastSequenceStartMillis = millis();
00117           }
00118         } else {
00119           switch (v_direction) {
00120             case CLOCKWISE:
00121               if (v_state == c_signalSequenceCW[v_sequenceStep]) {
00122                 v_sequenceStep++;
00123                 if (v_sequenceStep >= MAXSEQUENCESTEPS) { // Sequence has finished
00124                   result=v_direction;
00125                   v_lastResult=result;
00126                   v_direction=IDLE;
00127                   v_sequenceStep=0;
00128                 } else result=ACTIVE;
00129               } else {
00130                 // Invalid sequence
00131                 if (v_state == INITSTEP) { // Reset sequence in init state
00132                   v_direction=IDLE;
00133                   v_sequenceStep=0;
00134                 }
00135               }
00136               break;
00137             case COUNTERCLOCKWISE:
00138               if (v_state == c_signalSequenceCCW[v_sequenceStep]) {
00139                 v_sequenceStep++;
00140                 if (v_sequenceStep >= MAXSEQUENCESTEPS) { // Sequence has finished
00141                   result=v_direction;
00142                   v_lastResult=result;
00143                   v_direction=IDLE;
00144                   v_sequenceStep=0;
00145                 } else result=ACTIVE;
00146               } else {
00147                 // Invalid sequence
00148                 if (v_state == INITSTEP) { // Reset sequence in init state
00149                   v_direction=IDLE;
00150                   v_sequenceStep=0;
00151                 }
00152               }
00153               break;
00154           }
00155         }
00156         v_oldState = v_state;
00157       }
00158       // Prevent unsigned long overrun
00159       if (millis() - v_lastSequenceStartMillis > PREVENTSLEEPMS) {
00160           v_lastSequenceStartMillis = millis() - PREVENTSLEEPMS - 1;
00161       }
00162       return result;
00163     }
```

4.1.4.2 getAndResetLastRotation()

```
byte KY040::getAndResetLastRotation ( )  [inline]
```

Get and reset last finished rotation step (Do not use inside ISR)

Return values

KY040::CLOCKWISE	CLK/DT sequence for one step clockwise rotation has finished
KY040::COUNTERCLOCKWISE	CLK/DT sequence for one step counter-clockwise rotation has finished
KY040::IDLE	Rotary encoder is idle

Definition at line 172 of file KY040.h.

```
00173     {
00174       cli();
00175       byte result = v_lastResult;
00176       v_lastResult = IDLE;
00177       sei();
00178       return result;
00179     }
```

4.1.4.3 getRotation()

```
byte KY040::getRotation ( )  [inline]
```

Read and stores current pin state for CLK and DT and returns the current rotation state.

Reads pin state for CLK and DT with DigitalRead() and checks current rotation state by calling checkRotation()

Return values

KY040::CLOCKWISE	CLK/DT sequence for one step clockwise rotation has finished
KY040::COUNTERCLOCKWISE	CLK/DT sequence for one step counter-clockwise rotation has finished
KY040::IDLE	Rotary encoder is idle
KY040::ACTIVE	Rotary encoder is rotating, but the CLK/DT sequence has not finished

Definition at line 191 of file KY040.h.

```
00192     {
00193       setState((digitalRead(m_clk_pin)<<1)+digitalRead(m_dt_pin));
00194       return checkRotation();
00195     }
```

4.1.4.4 getState()

```
byte KY040::getState ( )  [inline]
```

Get stored pin states for CLK and DT (Left bit is for CLK, right bit is for DT). Should be called from ISR, when needed.

Returns

Stored pin states for CLK and DT in two bits (Left bit is for CLK, right bit is for DT)

Definition at line 202 of file KY040.h.

```
00203     {
00204       return v_state;
00205     }
```

4.1.4.5 readyForSleep()

```
bool KY040::readyForSleep ( )  [inline]
```

Checks, if it save to go to sleep.

Returns true, if device was running long enough to get a full sequence (Do not use inside ISR)

Return values

true	Yes, it is save to go to sleep
false	No, it is not save and you could miss signals, if you go to sleep anyway

Definition at line 215 of file KY040.h.

```
00216     {
00217       cli();
00218       unsigned long lastStepMillis = v_lastSequenceStartMillis;
00219       sei();
00220       return (millis()-lastStepMillis > PREVENTSLEEPMS);
00221     }
```

4.1.4.6 setState()

```
void KY040::setState (
            byte state )  [inline]
```

Stores pin states for CLK and DT (Left bit is for CLK, right bit is for DT). Should be called from ISR, when needed.

Parameters

in	*state*	Pin state for CLK and DT in two bits (Left bit is for CLK, right bit is for DT)

Definition at line 228 of file KY040.h.

```
00229     {
00230       v_state = state;
00231     }
```

The documentation for this class was generated from the following file:

- KY040.h

Chapter 5

File Documentation

5.1 pinChangeInterrupt.ino

```
00001 /*
00002  * Example for using the rotary encoder with pin change interrupts
00003  */
00004
00005 #include <KY040.h>
00006
00007 #define CLK_PIN 5 // aka. A
00008 #define DT_PIN 4 // aka. B
00009 KY040 g_rotaryEncoder(CLK_PIN,DT_PIN);
00010
00011 // Rotary encoder value (will be set in ISR)
00012 volatile int v_value=0;
00013
00014 // Enable pin change interrupt
00015 void pciSetup(byte pin) {
00016   *digitalPinToPCMSK(pin) |= bit (digitalPinToPCMSKbit(pin));  // enable pin
00017   PCIFR  |= bit (digitalPinToPCICRbit(pin)); // clear any outstanding interrupt
00018   PCICR  |= bit (digitalPinToPCICRbit(pin)); // enable interrupt for the group
00019 }
00020
00021 // ISR to handle pin change interrupt for D0 to D7 here
00022 ISR (PCINT2_vect) {
00023   // Process pin state
00024   switch (g_rotaryEncoder.getRotation()) {
00025     case KY040::CLOCKWISE:
00026       v_value++;
00027       break;
00028     case KY040::COUNTERCLOCKWISE:
00029       v_value--;
00030       break;
00031   }
00032 }
00033
00034 void setup() {
00035   Serial.begin(9600);
00036
00037   // Set pin change interrupt for CLK and DT
00038   pciSetup(CLK_PIN);
00039   pciSetup(DT_PIN);
00040 }
00041
00042 void loop() {
00043   static int lastValue = 0;
00044   int value;
00045
00046   // Get rotary encoder value set in ISR
00047   cli();
00048   value = v_value;
00049   sei();
00050
00051   // Show, if value has changed
00052   if (lastValue != value) {
00053     Serial.println(value);
00054     lastValue = value;
00055   }
00056 }
```

5.2 pinChangeInterruptDualEncoders.ino

```
00001 /*
00002  * Example for using two rotary encoders with pin change interrupts
00003  */
00004
00005 #include <KY040.h>
00006
00007 // First rotary encoder
00008 #define X_CLK_PIN 5 // aka. A
00009 #define X_DT_PIN 4 // aka. B
00010 KY040 g_rotaryEncoderX(X_CLK_PIN,X_DT_PIN);
00011
00012 // Second rotary encoder
00013 #define Y_CLK_PIN 7 // aka. A
00014 #define Y_DT_PIN 6 // aka. B
00015 KY040 g_rotaryEncoderY(Y_CLK_PIN,Y_DT_PIN);
00016
00017 // Rotary encoder values (will be set in ISR)
00018 volatile int v_valueX=0;
00019 volatile int v_valueY=0;
00020
00021 // Enable pin change interrupt
00022 void pciSetup(byte pin) {
00023   *digitalPinToPCMSK(pin) |= bit (digitalPinToPCMSKbit(pin));  // enable pin
00024   PCIFR  |= bit (digitalPinToPCICRbit(pin)); // clear any outstanding interrupt
00025   PCICR  |= bit (digitalPinToPCICRbit(pin)); // enable interrupt for the group
00026 }
00027
00028 // ISR to handle pin change interrupts for D0 to D7 here
00029 ISR (PCINT2_vect) {
00030   // Read pin states with PIND (Faster replacement for digitalRead, better for fast interrupts, but
      harder to read)
00031   byte state = PIND;
00032   byte stateX = ((state & 0b00110000)»4);
00033   byte stateY = ((state & 0b11000000)»6);
00034
00035   // Process first rotary encoder
00036   g_rotaryEncoderX.setState(stateX); // Store CLK/DT states
00037   // Process stored state
00038   switch (g_rotaryEncoderX.checkRotation()) {
00039     case KY040::CLOCKWISE:
00040       v_valueX++;
00041       break;
00042     case KY040::COUNTERCLOCKWISE:
00043       v_valueX--;
00044       break;
00045   }
00046
00047   // Process second rotary encoder
00048   g_rotaryEncoderY.setState(stateY); // Store CLK/DT states
00049   // Process stored state
00050   switch (g_rotaryEncoderY.checkRotation()) {
00051     case KY040::CLOCKWISE:
00052       v_valueY++;
00053       break;
00054     case KY040::COUNTERCLOCKWISE:
00055       v_valueY--;
00056       break;
00057   }
00058 }
00059
00060 void setup() {
00061   Serial.begin(9600);
00062
00063   // Set pin change interrupt for CLK and DT
00064   pciSetup(X_CLK_PIN);
00065   pciSetup(X_DT_PIN);
00066   pciSetup(Y_CLK_PIN);
00067   pciSetup(Y_DT_PIN);
00068 }
00069
00070 void loop() {
00071   static int lastValueX = 0;
00072   static int lastValueY = 0;
00073
00074   int valueX, valueY;
00075
00076   // Get rotary encoder values set in ISR
00077   cli();
00078   valueX = v_valueX;
00079   valueY = v_valueY;
00080   sei();
00081
00082   // Show, if value has changed
00083   if ((lastValueX != valueX) || (lastValueY != valueY)) {
00084     Serial.print("X:");
```

```
00085     Serial.print(valueX);
00086     Serial.print(" Y:");
00087     Serial.println(valueY);
00088     lastValueX = valueX;
00089     lastValueY = valueY;
00090   }
00091 }
```

5.3 pinChangeInterruptPowerSave.ino

```
00001 /*
00002  * Example for using the rotary encoder with pin change interrupts and
00003  * SLEEP_MODE_PWR_SAVE sleep mode
00004  */
00005
00006 #include <avr/sleep.h>
00007 #include <KY040.h>
00008
00009 #define CLK_PIN 5 // aka. A
00010 #define DT_PIN 4 // aka. B
00011 KY040 g_rotaryEncoder(CLK_PIN,DT_PIN);
00012
00013 // Rotary encoder value (will be set in ISR)
00014 volatile int v_value=0;
00015
00016 // Enable pin change interrupt
00017 void pciSetup(byte pin) {
00018   *digitalPinToPCMSK(pin) |= bit (digitalPinToPCMSKbit(pin));  // enable pin
00019   PCIFR  |= bit (digitalPinToPCICRbit(pin)); // clear any outstanding interrupt
00020   PCICR  |= bit (digitalPinToPCICRbit(pin)); // enable interrupt for the group
00021 }
00022
00023 // ISR to handle pin change interrupt for D0 to D7 here
00024 ISR (PCINT2_vect) {
00025   // Faster replacement for digitalRead, better for interrupts, but harder to read
00026   byte state = ((PIND & 0b00110000)»4);
00027   g_rotaryEncoder.setState(state); // Store CLK/DT states
00028   // Process stored state
00029   switch (g_rotaryEncoder.checkRotation()) {
00030     case KY040::CLOCKWISE:
00031       v_value++;
00032       break;
00033     case KY040::COUNTERCLOCKWISE:
00034       v_value--;
00035       break;
00036   }
00037 }
00038
00039 void setup() {
00040   Serial.begin(9600);
00041
00042   // Set pin change interrupt for CLK and DT
00043   pciSetup(CLK_PIN);
00044   pciSetup(DT_PIN);
00045
00046   // Set sleep mode to SLEEP_MODE_PWR_SAVE
00047   set_sleep_mode(SLEEP_MODE_PWR_SAVE);
00048 }
00049
00050 void loop() {
00051   static int lastValue = 0;
00052   int value;
00053
00054   // Go to sleep when rotary encoder has no rotation for ~150 milliseconds
00055   if (g_rotaryEncoder.readyForSleep()) sleep_mode();
00056
00057   // Get rotary encoder value set in ISR
00058   cli();
00059   value = v_value;
00060   sei();
00061
00062   // Show, if value has changed
00063   if (lastValue != value) {
00064     Serial.println(value);
00065     Serial.flush();
00066     lastValue = value;
00067   }
00068 }
```

5.4 pollingNoInterrupts.ino

```
00001 /*
00002  * Example for using the rotary encoder without interrupts
00003  * in polling mode
00004  */
00005
00006 #include <KY040.h>
00007
00008 #define CLK_PIN 5 // aka. A
00009 #define DT_PIN 4 // aka. B
00010 KY040 g_rotaryEncoder(CLK_PIN,DT_PIN);
00011
00012 void setup() {
00013   Serial.begin(9600);
00014   // If your rotary encoder has no builtin pullup resistors for CLK (aka. A) and DT (aka. B) uncomment
      the following two lines
00015   // pinMode(CLK_PIN,INPUT_PULLUP);
00016   // pinMode(DT_PIN,INPUT_PULLUP);
00017 }
00018
00019 void loop() {
00020   static int value=0;
00021
00022   // You have to run getRotation() very frequently in loop to prevent missing rotary encoder signals
00023   // If this is not possible take a look at the pinChangeInterrupt examples
00024   switch (g_rotaryEncoder.getRotation()) {
00025     case KY040::CLOCKWISE:
00026       value++;
00027       Serial.println(value);
00028       break;
00029     case KY040::COUNTERCLOCKWISE:
00030       value--;
00031       Serial.println(value);
00032       break;
00033   }
00034 }
```

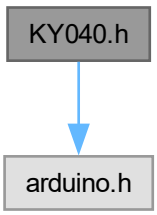
5.5 withInterrupt.ino

```
00001 /*
00002  * Example for using a rotary encoders with interrupts
00003  */
00004
00005 #include <KY040.h>
00006
00007 // Rotary encoder
00008 #define CLK_PIN 3 // aka. A
00009 #define DT_PIN 2 // aka. B
00010 KY040 g_rotaryEncoder(CLK_PIN,DT_PIN);
00011
00012 // Rotary encoder value (will be set in ISR)
00013 volatile int v_value=0;
00014
00015 // ISR to handle the interrupts for CLK and DT
00016 void ISR_rotaryEncoder() {
00017   // Process pin states for CLK and DT
00018   switch (g_rotaryEncoder.getRotation()) {
00019     case KY040::CLOCKWISE:
00020       v_value++;
00021       break;
00022     case KY040::COUNTERCLOCKWISE:
00023       v_value--;
00024       break;
00025   }
00026 }
00027
00028
00029 void setup() {
00030   Serial.begin(9600);
00031
00032   // Set interrupts for CLK and DT
00033   attachInterrupt(digitalPinToInterrupt(CLK_PIN), ISR_rotaryEncoder, CHANGE);
00034   attachInterrupt(digitalPinToInterrupt(DT_PIN), ISR_rotaryEncoder, CHANGE);
00035 }
00036
00037 void loop() {
00038   static int lastValue = 0;
00039   int value;
00040
00041   // Get rotary encoder value set in ISR
00042   cli();
00043   value = v_value;
```

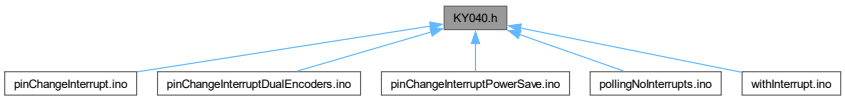
```
00044   sei();
00045
00046   // Show, if value has changed
00047   if (lastValue != value) {
00048     Serial.println(value);
00049     lastValue = value;
00050   }
00051 }
```

5.6 KY040.h File Reference

`#include <arduino.h>`
Include dependency graph for KY040.h:



This graph shows which files directly or indirectly include this file:



Classes

- class KY040

Macros

- #define KY040_VERSION "1.0.1"
- #define PREVENTSLEEPMS 150
- #define INITSTEP 0b11
- #define MAXSEQUENCESTEPS 4

5.6.1 Detailed Description

Class: KY040

Description: Class for KY-040 rotary encoders. Without builtin pull up resistors for CLK/DT you have to set pinMode(,INPUT_PULLUP) before using the class. The class works with or without interrupts and prevents bounces by ignoring invalid CLK/DT sequences

License: 2-Clause BSD License Copyright (c) 2024 codingABI For details see: LICENSE.txt

Valid clockwise sequence for CLK/DT: Low/High->Low/Low->High/Low->High/High

```
       0 1 2 3
    --+   +----   High
CLK   |   |
      +---+       Low

    ----+   +--   High
DT      |   |
        +---+     Low
```

Valid counter-clockwise sequence for CLK/DT: High/Low->Low/Low->Low/High->High/High

```
       0 1 2 3
    ----+   +---  High
CLK     |   |
        +---+     Low

    --+   +-----  High
DT    |   |
      +---+       Low
```

Home: https://github.com/codingABI/KY040

Author

codingABI https://github.com/codingABI/

Copyright

2-Clause BSD License

Version

1.0.1

Definition in file KY040.h.

5.6.2 Macro Definition Documentation

5.6.2.1 INITSTEP

```
#define INITSTEP 0b11
```

Definition at line 57 of file KY040.h.

5.6.2.2 KY040_VERSION

```
#define KY040_VERSION "1.0.1"
```

Library version

Definition at line 50 of file KY040.h.

5.6.2.3 MAXSEQUENCESTEPS

```
#define MAXSEQUENCESTEPS 4
```

Definition at line 59 of file KY040.h.

5.6.2.4 PREVENTSLEEPMS

```
#define PREVENTSLEEPMS 150
```

When using sleep modes wait X milliseconds for next sleep after a CLK/DT sequence start do prevent missing signals

Definition at line 55 of file KY040.h.

5.7 KY040.h

Go to the documentation of this file.

```
00001
00047 #pragma once
00048
00050 #define KY040_VERSION "1.0.1"
00051
00052 #include <arduino.h>
00053
00055 #define PREVENTSLEEPMS 150
00056 // Pin idle state
00057 #define INITSTEP 0b11
00058 // Max steps for a signal sequence
00059 #define MAXSEQUENCESTEPS 4
00060
00062 class KY040 {
00063   public:
00065     enum directions
00066     {
00067       IDLE,
00068       ACTIVE,
00069       CLOCKWISE,
00070       COUNTERCLOCKWISE
00071     };
00072
00079     KY040(byte clk_pin, byte dt_pin)
00080     {
00081       m_clk_pin = clk_pin; // aka. A
00082       m_dt_pin = dt_pin; // aka. B
00083       v_state = 255;
00084       v_lastResult = IDLE;
00085       v_lastSequenceStartMillis = millis();
00086       v_sequenceStep = 0;
00087       v_direction = IDLE;
00088       v_oldState = INITSTEP;
00089     }
00090
00102     byte checkRotation()
00103     {
00104       byte result = IDLE;
00105
```

```
00106       if (v_state != v_oldState) { // State changed?
00107         if (v_sequenceStep == 0) { // Check for begin of rotation
00108           if (v_state == c_signalSequenceCW[0]) { // Begin of CW
00109             v_direction=CLOCKWISE;
00110             v_sequenceStep = 1;
00111             v_lastSequenceStartMillis = millis();
00112           }
00113           if (v_state == c_signalSequenceCCW[0]) { // Begin of CCW
00114             v_direction=COUNTERCLOCKWISE;
00115             v_sequenceStep = 1;
00116             v_lastSequenceStartMillis = millis();
00117           }
00118         } else {
00119           switch (v_direction) {
00120             case CLOCKWISE:
00121               if (v_state == c_signalSequenceCW[v_sequenceStep]) {
00122                 v_sequenceStep++;
00123                 if (v_sequenceStep >= MAXSEQUENCESTEPS) { // Sequence has finished
00124                   result=v_direction;
00125                   v_lastResult=result;
00126                   v_direction=IDLE;
00127                   v_sequenceStep=0;
00128                 } else result=ACTIVE;
00129               } else {
00130                 // Invalid sequence
00131                 if (v_state == INITSTEP) { // Reset sequence in init state
00132                   v_direction=IDLE;
00133                   v_sequenceStep=0;
00134                 }
00135               }
00136               break;
00137             case COUNTERCLOCKWISE:
00138               if (v_state == c_signalSequenceCCW[v_sequenceStep]) {
00139                 v_sequenceStep++;
00140                 if (v_sequenceStep >= MAXSEQUENCESTEPS) { // Sequence has finished
00141                   result=v_direction;
00142                   v_lastResult=result;
00143                   v_direction=IDLE;
00144                   v_sequenceStep=0;
00145                 } else result=ACTIVE;
00146               } else {
00147                 // Invalid sequence
00148                 if (v_state == INITSTEP) { // Reset sequence in init state
00149                   v_direction=IDLE;
00150                   v_sequenceStep=0;
00151                 }
00152               }
00153               break;
00154           }
00155         }
00156         v_oldState = v_state;
00157       }
00158       // Prevent unsigned long overrun
00159       if (millis() - v_lastSequenceStartMillis > PREVENTSLEEPMS) {
00160           v_lastSequenceStartMillis = millis() - PREVENTSLEEPMS - 1;
00161       }
00162       return result;
00163     }
00164
00172     byte getAndResetLastRotation()
00173     {
00174       cli();
00175       byte result = v_lastResult;
00176       v_lastResult = IDLE;
00177       sei();
00178       return result;
00179     }
00180
00191     byte getRotation()
00192     {
00193       setState((digitalRead(m_clk_pin)<<1)+digitalRead(m_dt_pin));
00194       return checkRotation();
00195     }
00196
00202     byte getState()
00203     {
00204       return v_state;
00205     }
00206
00215     bool readyForSleep()
00216     {
00217       cli();
00218       unsigned long lastStepMillis = v_lastSequenceStartMillis;
00219       sei();
00220       return (millis()-lastStepMillis > PREVENTSLEEPMS);
00221     }
00222
```

```
00228     void setState(byte state)
00229     {
00230       v_state = state;
00231     }
00232   private:
00233     byte m_clk_pin; // aka. A
00234     byte m_dt_pin; // aka. B
00235     volatile byte v_state;
00236     volatile byte v_lastResult;
00237     volatile unsigned long v_lastSequenceStartMillis;
00238     volatile byte v_sequenceStep;
00239     volatile byte v_direction;
00240     volatile byte v_oldState;
00241     // CLK/DT sequence for a clockwise rotation (One byte instead of a byte array would be enough for
      the four 2-bit values, but are harder to read)
00242     const byte c_signalSequenceCW[MAXSEQUENCESTEPS] = {0b01,0b00,0b10,INITSTEP};
00243     // CLK/DT sequence for a counter-clockwise rotation (One byte instead of a byte array would be
      enough for the four 2-bit values, but are harder to read)
00244     const byte c_signalSequenceCCW[MAXSEQUENCESTEPS] = {0b10,0b00,0b01,INITSTEP};
00245
00246 };
```

Index