Arduino-AD9850-DDS

Der AD9850 (Datenblatt) ist ein DDS, der ein Sinussignal von 0-40MHz erzeugen kann.
In diesem Projekt wird er von einem Arduino Nano angesteuert.
Die Frequenzeinstellung sowie die Quartz-Kalibration erfolgt mit einem Drehencoder und einem I2C-Textdisplay (16×2). Die eingestellten Werte können im Arduino-EEPROM gespeichert werden.

Verdrahtung am Arduino		
Modul	**Pin Name**	**Arduino Pin**
I2C-Textdisplay	SDA	A4
	SCL	A5
DrehEncoder	CLK	D2
	DT	D3
(Button)	SW	D4 (mit 10kOhm zu VCC)
AD9850	W_CLK	D5
	FU_UD/FQ_UD	D6
	Data	D7
	Reset	D8

Benutzte Bibliotheken:

- Drehencoder: Encoder http://www.pjrc.com/teensy/td_libs_Encoder.html LLC - Paul Stoffregen paul@pjrc.com
- EEPROMex http://thijs.elenbaas.net/2012/07/extended-eeprom-library-for-arduino/ Thijs Elenbaas, GNU LGPL
- zum Teil integriert (muss nicht installiert werden, hier nur als Referenz) AD9850 https://github.com/F4GOJ/AD9850 Created 23/08/2014, Christophe Caiveau f4goj@free.fr, Public Domain

Kurze Warnung vorab: ich bin kein Programmierer und es gibt die ein oder andere Stelle, die man noch verbessern könnte*. Auch wenn der Code gut funktioniert, bin ich für jeden Tip per Mail an danielwf@hackerspace-bremen.de dankbar - man lernt ja schließlich nie aus ;)

**Todo:* Serielle Frequenzeingabe; DrehEncoder-Eingabe tlw. ungenau; EEPROM-Update wird immer geschrieben, wenn der Mode geändert wird → Nur schreiben, wenn sich der Wert im EEPROM auch wirklich geändert hat.

Das gesamte Projekt inkl. Libraries kann hier als zip-Archiv herunterladen werden.

Arduino-Sketch Arduino-AD9850-DDS.ino (Arduino 1.6.9, Stand 17.11.2016)

```
//
//    .-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------.
//    |
Arduino-AD9850-DDS
|
//    '-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------'
//     by Daniel Wendt-Fröhlich, DL2AB (danielwf@hackerspace-bremen.de,
dl2ab@darc.de) for
//          "Hackerspace Bremen e.V." https://hackerspace-bremen.de / HSHB
Amateur Radio Group http://hshb.de/afu
//     License CC-by-SA 3.0 - Nov 2016 - Bremen(GER) --
http://creativecommons.org/licenses/by-sa/3.0/de/
//
//                                      Frequency selectable with push button
rotary encoder
//                                       incl. XTAL-calibration and saving
values to EEPROM
//     Used Libraries:
//     Encoder http://www.pjrc.com/teensy/td_libs_Encoder.html PJRC.COM, LLC
- Paul Stoffregen <paul@pjrc.com>
//     EEPROMex
http://thijs.elenbaas.net/2012/07/extended-eeprom-library-for-arduino/
Thijs Elenbaas, GNU LGPL
//     in parts AD9850 https://github.com/F4GOJ/AD9850  Created 23/08/2014,
Christophe Caiveau f4goj@free.fr, Public Domain
//     NewLiquidCrystal
https://bitbucket.org/fmalpartida/new-liquidcrystal/wiki/Home see website
for authors and license
//
//
//     todo: set frequency via serial interface, better rotary encoder
detection, EEPROM-Update only if values have really changed.
```

```
//
//     ------------------------------------------------Connections----------
-----------------------------------------------
//        Display SDA: A4 (check/modify 'LiquidCrystal_I2C lcd' for your
used I2C-LCD-Adapter, I2C-Adress, PinOut)
//        Display SCL: A5
//
//         Rotary CLK: D2 (check/modify encResolution for your rotary
encoder)
//         Rotary  DT: D3
//  Button/Rotary  SW: D4 (with 10k-PullUp to VCC)
//
//       AD9850 W_CLK: D5
//       AD9850 FU_UD: D6
//       AD9850  Data: D7
//       AD9850 Reset: D8
//
//    .-------------------------------------------------------------------------
-------------------------------------------------------------------------------
---------------.
//    |                                                                Libraries
and Settings
|
//    '-------------------------------------------------------------------------
-------------------------------------------------------------------------------
---------------'
                                                                             //
Frequencies used when no stored values are found (or are 0)
double ddsFreq = 14070000;                                                   // DDS
Standard frequency
double calibFreq = 125000000;                                                // XTAL
Standard frequency
int ddsPhase = 0;                                                            //
Phase for DDS (not further used in this project, but need for DDS)

double frequency;                                                            //
working values for LCD and modification...
double newfrequency;                                                         //
...will be set in setup
byte freqCursor = 0;
byte Mode = 0;                                                               // Mode
0=DDS 1=Calibrate


//     ------------------------------------------------EEPROM for Double-
Values---------------------------------------
#include <EEPROMex.h>                                                        //
EEPROM-Libs for more simple Double-Handling,
http://thijs.elenbaas.net/2012/07/extended-eeprom-library-for-arduino/
#include <EEPROMVar.h>
const int eepadrDDS = 10;                                                    //
```

```
Adresses in EEPROM
const int eepadrCAL = 20;
double eepRead = 0;                                                    // read
value from EEPROM


//     ------------------------------------------------Display-------------
-----------------------------------------
#include <Wire.h>
#include <LiquidCrystal_I2C.h>                                         // I2C-
LCD-Library, included in Arduino-IDE
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);        // Set
the LCD I2C address, SainSmartLCD2004 0x3F, maybe try 0x38 or 0x20 or 0x27
bool DisplayRefresh = 1;


//     ------------------------------------------------Encoder-------------
-----------------------------------------
#include <Encoder.h>                                                   // Encoder
Library using interupts, http://www.pjrc.com/teensy/td_libs_Encoder.html
Encoder myEnc(2, 3);                                                   // Pins 2+3
are usable for Interupts on the Arduino Nano
byte oldPosition  = 0;                                                 // beginning
position of rotary counter, must be 0
byte encResolution = 4;                                                // counted
steps per rotary-step
int pushButton = 4;                                                    // Button is
connected to Pin 4
unsigned long rotaryDebounce = millis() + 100;                         // removes
wrong detected/counted movements
unsigned long buttonDebounce = millis() + 300;                         // same for
button (slows down ;)
byte buttonHold = 0;

int8_t rotaryDirection(){                                              // returns
Direction of rotary encoder with -1, 0 oder 1
  byte i = 0;                                                          // default =
no rotation
  byte newPosition = myEnc.read();                                     // reads
position of encoder-lib
  if (  newPosition != oldPosition) {                                  // if
position has changed
    if ( (newPosition % encResolution == 0) && (millis() >= rotaryDebounce)
) { // calculate real steps according to rotary-resolution and debouncing
      if (oldPosition > newPosition) i = -1;                           // return
value
      if (oldPosition < newPosition) i = 1;
      rotaryDebounce = millis() + 100;                                 // timer for
debouncing
    }
```

```
    oldPosition = newPosition;                                    // working
value for calculating further rotations
  }
  return i;
}

//     ------------------------------------------------AD9850---------------
-----------------------------------------

                                                                  // values
and function from https://github.com/F4GOJ/AD9850 -- lib cannot be used,
"DATA" is also in I2C-LCD-Lib
const int adPinWCLK = 5;
const int adPinFQUD = 6;
const int adPinDATA = 7;
const int adPinRESET = 8;
uint32_t adDeltaphase;                                            //
calculated value send to AD9850
uint8_t adPhase;

void adSetfreq(double f, uint8_t p) {                             // function
gets frequency and phase
  adDeltaphase = f * 4294967296.0 / calibFreq;                    //
calculation for value to send
  adPhase = p << 3;                                               // bitshift
for phase
  for (int i=0; i<4; i++, adDeltaphase>>=8) {                     // shift out
the double (=4bytes) via DDS-pins
    shiftOut(adPinDATA, adPinWCLK, LSBFIRST, adDeltaphase & 0xFF);
  }
  shiftOut(adPinDATA, adPinWCLK, LSBFIRST, adPhase & 0xFF);       // shift out
phase-value
  digitalWrite(adPinFQUD, HIGH); digitalWrite(adPinFQUD, LOW);  // DDS sets
to sent value after FQUD is up
}




//    .-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------.
//    |                                                               SETUP
|
//    '-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------'

void setup() {

  pinMode(pushButton, INPUT);                            // set pushButton as
```

```
input

  Serial.begin(9600);                                  // setup serial
connection

  lcd.begin(16,2);                                     // initialize the 16x2-
lcd, backlight is lit
  lcd.backlight();                                     // switch backlight on
  lcd.setCursor(3,0); lcd.print("Arduino-");           // set display-cursor x,y
and print text
  lcd.setCursor(2,1); lcd.print("AD9850-DDS");
  delay(2000);

  lcd.noBacklight();                                   // switch backlight off,
no visible screen transistion, better short-term readablity
  lcd.clear();                                         // clear content on
display
  lcd.setCursor(0,0); lcd.print("short: sel.Pos.");  // Help-Message
  lcd.setCursor(0,1); lcd.print("hold:save f/cal");
  delay(300);
  lcd.backlight();
  delay(2000);
                                                       // ------------------
EEPROM Reading while Help-Message is still on the display------------------
------------
  eepRead = EEPROM.readDouble(eepadrDDS);              // read double from
eeprom for DDS-frequency
  if (eepRead > 0) ddsFreq = eepRead;                  // check if empty
(important for new installations) if not, overwrite with stored value
  eepRead = EEPROM.readDouble(eepadrCAL);              // same for Xtal...
  if (eepRead > 0) calibFreq = eepRead;
  frequency = ddsFreq;                                 // set working values for
LCD and modification
  newfrequency = frequency;

  lcd.noBacklight();
  lcd.clear();
  lcd.setCursor(0,0); lcd.print("DDS          _");
  lcd.setCursor(0,1); lcd.print("Freq 00000000 Hz");
  delay(300);
  lcd.backlight();

                                                       // ------------------
Initialize the AD9850-DDS -------------------------------
  pinMode(adPinWCLK, OUTPUT);                             // set output-pins
  pinMode(adPinFQUD, OUTPUT);
  pinMode(adPinDATA, OUTPUT);
  pinMode(adPinRESET, OUTPUT);
  digitalWrite(adPinRESET, HIGH); digitalWrite(adPinRESET, LOW);  // init
the DDS
```

```
  digitalWrite(adPinWCLK, HIGH); digitalWrite(adPinWCLK, LOW);
  digitalWrite(adPinFQUD, HIGH); digitalWrite(adPinFQUD, LOW);

  adSetfreq(ddsFreq, ddsPhase);                    // send values to AD9850

}

//    .-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------.
//    |                                                                LOOP
|
//    '-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------'


void loop() {

    int8_t rotaryValue = rotaryDirection();
// value for last encoder rotation (-1,0,+1)


// ------------------Button-----------------------------------------------
    if (digitalRead(pushButton)) buttonHold = 0;
// set Button-Hold-Counter to 0, if button is released
    if (buttonHold >= 10) {
// if Button-Hold-Counter is 6, set Mode to  1 (Calibration)
      EEPROM.writeDouble(eepadrDDS, ddsFreq);
// Writes values to EEPROM
      EEPROM.writeDouble(eepadrCAL, calibFreq);
      Mode++;
// set next Mode
      if (Mode == 2) Mode = 0;
// There are only modes 0 + 1
      buttonHold = 0;
// reset counter for button hold
      lcd.clear();
      DisplayRefresh = 1;
// force rewriting of lcd after mode changed
      if (Mode == 0) frequency = ddsFreq;
// set the working frequency related to the mode
      if (Mode == 1) frequency = calibFreq;
    }

    if ( (!digitalRead(pushButton)) && (millis() >= buttonDebounce) ) {
// if button is pressed, debounce-timer (and timer für ButtonHold-Counter)
      freqCursor++;
// set the cursor to next position
      if (freqCursor == 8) freqCursor = 0;
// reset cursor if it is bigger than 8
```

```
      DisplayRefresh = 1;
// force rewriting of lcd after something changed
      // lcd.setCursor((12-freqCursor),0); lcd.print("_");
// Cursor-Position on Display
      buttonDebounce = millis() + 300;
// Debounce-Timer
      buttonHold++;
// Button-Hold-cunter
    }


// ------------------rotary encoder-----------------------------------------
-----
    if (freqCursor == 0) newfrequency = frequency + (rotaryValue);
// calculates the frequency using Cursor-Position and rotary-encoder
    if (freqCursor == 1) newfrequency = frequency + (rotaryValue * 10);
// the simple way works...
    if (freqCursor == 2) newfrequency = frequency + (rotaryValue * 100);
    if (freqCursor == 3) newfrequency = frequency + (rotaryValue * 1000);
    if (freqCursor == 4) newfrequency = frequency + (rotaryValue * 10000);
    if (freqCursor == 5) newfrequency = frequency + (rotaryValue * 100000);
    if (freqCursor == 6) newfrequency = frequency + (rotaryValue * 1000000);
    if (freqCursor == 7) newfrequency = frequency + (rotaryValue *
10000000);
    if (newfrequency <= 0) newfrequency = 0;
    if (Mode == 0) {
      if (newfrequency <= 0) newfrequency = 0;
// frequency 0-40M with Mode 0
      if (newfrequency >= 40000000) newfrequency = 40000000;
    }

    // Print frequency
// ------------------Display Frequency--------------------------------------
-----------
    if ( (newfrequency != frequency) || DisplayRefresh ) {
// Only changes the display after frequency-adjustment or DisplayRefresh-Bit
      DisplayRefresh = 0;
      frequency = newfrequency;
// changes the real frequency
      byte freqSize = 0;
// calculates the display size of the frequency
      double freqDec = 10;
      while (freqDec <= frequency) {
        freqSize++;
        freqDec = freqDec * 10;
      }
      if (Mode == 0) { lcd.setCursor(0,0); lcd.print("DDS          "); }
// 1st line on lcd...
      if (Mode == 1) { lcd.setCursor(0,0); lcd.print("Cal          "); }
//          ...related to mode
```

```
      lcd.setCursor((12-freqCursor),0); lcd.print("_");
// Cursor-Position on Display
      if (Mode == 0) { lcd.setCursor(0,1); lcd.print("Freq 00000000 Hz"); }
// 2nd line on lcd...
      if (Mode == 1) { lcd.setCursor(0,1); lcd.print("Xtl 000000000 Hz"); }
//          ...related to mode
      lcd.setCursor( (12-freqSize) ,1);
// set cursor with calculated display size of frequency
      lcd.print(frequency,0);
// print the frequency
      Serial.println(frequency,0);
// serial output of the frequency
      if (Mode == 0) ddsFreq = frequency;
      if (Mode == 1) calibFreq = frequency;
      adSetfreq(ddsFreq, ddsPhase);
// send values to AD9850
    }

}
```