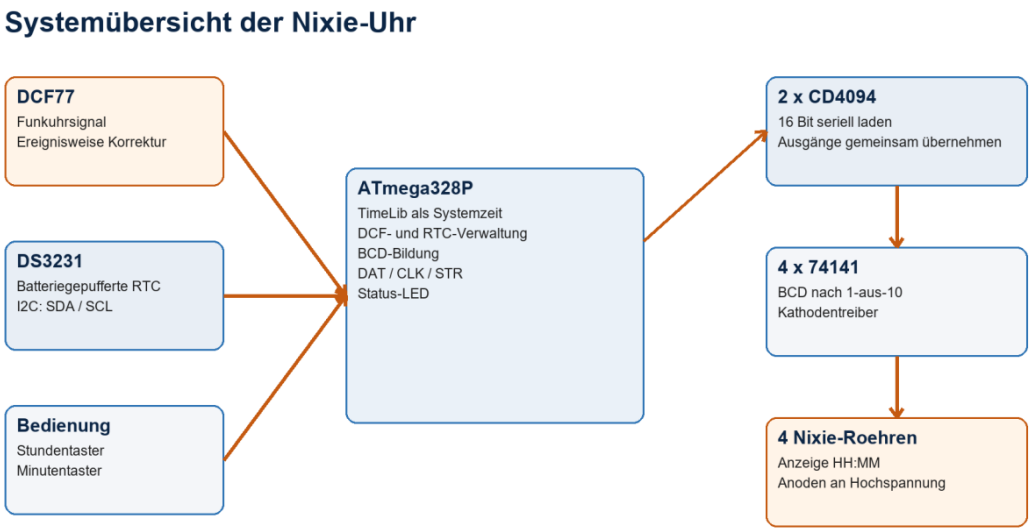
TECHNISCHE PROGRAMMDOKUMENTATION

DCF_Clock_V4a

Steuerung einer DCF77-synchronisierten Nixie-Uhr

Vom Zeittelegramm bis zur sichtbaren Anzeige HH:MM



Systemkontext des Steuerprogramms

Programmversion 4a | Dokumentationsstand 04.08.2026

Autor des Steuerprogramms: Thomas F. Liebe

Inhaltsübersicht

1 Zweck, Zielgruppe und Abgrenzung

2 Gesamtsystem und Softwarearchitektur

3 Zeitmodell und Priorität der Zeitquellen

4 Hardwarebezug, Pins und Bibliotheken

5 Programmstart und Selbsttest

6 Erzeugung der Nixie-Anzeigedaten

7 Ablauf der Hauptschleife

8 Bedienung und Statusanzeige

9 Funktionen und globale Zustände im Detail

10 Typische Betriebssituationen und Grenzfälle

11 Übersetzung, Diagnose und Inbetriebnahme

12 Systemgrenzen, Wartung und Erweiterbarkeit

13 Prüfvorschläge für Hardware und Software

14 Begriffe und Dokumentationsbasis

Leseführung: Kapitel 2 bis 8 erklären den Signal- und Programmablauf. Kapitel 9 dient als Funktionsreferenz; Kapitel 10 bis 13 unterstützen Inbetriebnahme, Fehlersuche und spätere Erweiterungen.

1 Zweck, Zielgruppe und Abgrenzung

Dieses Dokument erläutert die Funktionsweise des Arduino-Sketches DCF_Clock_V4a.ino in Verbindung mit der vorhandenen Nixie-Uhr. Es richtet sich an erfahrene Elektronikbastler sowie an angehende Studierende der Elektronik, die digitale Schaltungen lesen können und über Grundkenntnisse in C/C++ beziehungsweise der Arduino-Umgebung verfügen.

Im Mittelpunkt steht nicht die allgemeine Geschichte der Nixie-Röhre, sondern die konkrete Umsetzung: Wie gewinnt das Programm eine verlässliche Uhrzeit, wie werden vier Dezimalziffern in die Bitbelegung zweier CD4094 umgesetzt, wie gelangen diese Bits zu den 74141-Kathodentreibern und wie reagiert die Software auf DCF77-Empfang, RTC-Zustand und Tasterbetätigung?

Maßgeblich ist der Programmstand vom 3. August 2026. Die ältere Softwareprüfung wird als Entwicklungshistorie herangezogen; ihre damaligen Kritikpunkte beschreiben nicht mehr automatisch den aktuellen Zustand. Die inzwischen umgesetzten Korrekturen sind in dieser Dokumentation als reguläre Programmlogik dargestellt.

Sicherheit: Nixie-Schaltungen arbeiten mit berührungsgefährlicher Hochspannung. Die Softwarebeschreibung ersetzt weder Schutzmaßnahmen noch eine fachgerechte Prüfung des Hochspannungsteils. Messungen dürfen nur mit geeigneter Ausrüstung und ausreichender Qualifikation erfolgen.

2 Gesamtsystem und Softwarearchitektur

2.1 Aufgabe des Steuerprogramms

Das Steuerprogramm verbindet drei Eingabebereiche mit zwei Ausgabebereichen. Eingaben sind das DCF77-Signal, die DS3231-Echtzeituhr und zwei Taster. Ausgaben sind die vierstellige Nixie-Anzeige sowie eine Status-LED. Dazwischen verwaltet der ATmega328P eine interne Systemzeit und übersetzt Stunde und Minute in die zur Leiterplatte passende Bitfolge.

- DCF77 liefert bei einem vollständig erkannten Telegramm eine absolute Zeitkorrektur.
- Der DS3231 bewahrt Datum und Uhrzeit bei ausgeschalteter Hauptversorgung und dient im laufenden Betrieb als stabile Referenz.
- TimeLib stellt die aktive Systemzeit bereit, auf die der übrige Programmablauf zugreift.
- Die Taster ändern Stunde oder Minute unmittelbar und schreiben den neuen Wert auch in eine vorhandene RTC.
- CD4094 und 74141 bilden die Schnittstelle zwischen den wenigen Mikrocontroller-Pins und den vier Nixie-Röhren.

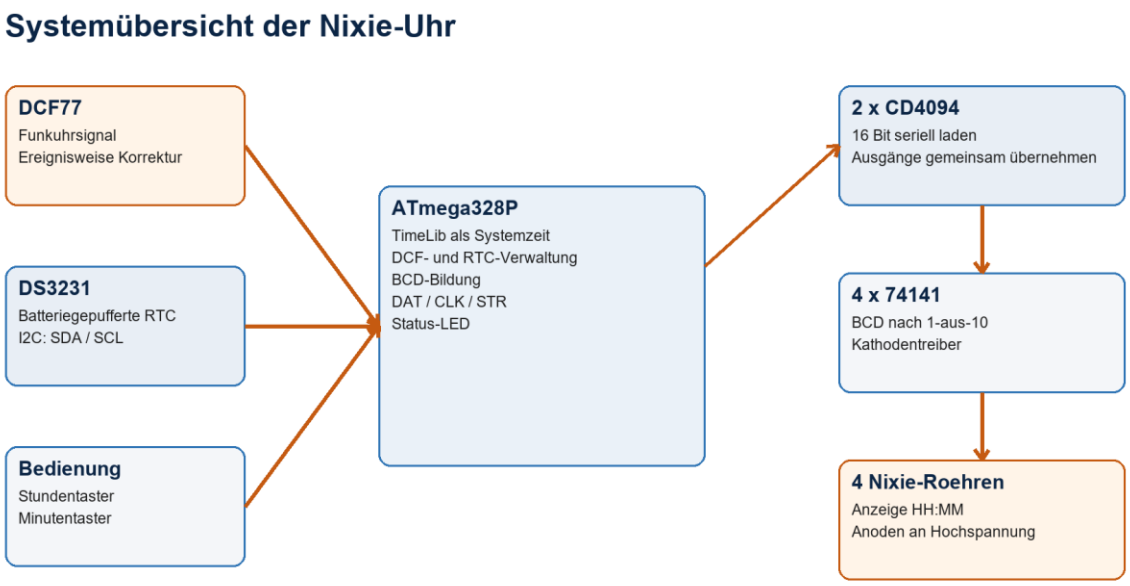


Abbildung 1: Funktionsblöcke und Informationsfluss

2.2 Trennung der Softwareebenen

Die Programmlogik lässt sich in vier Ebenen gliedern. Die Bibliotheken kapseln hardwarenahe Einzelheiten wie DCF-Decodierung, I2C-Kommunikation und Tasterentprellung. Die Zustandsvariablen beschreiben, welche Zeitquellen vorhanden und gültig sind. setup() stellt einen definierten Ausgangszustand her. loop() verarbeitet anschließend Ereignisse und aktualisiert die Anzeige höchstens einmal pro Sekunde.

Ebene	Aufgabe	Beispiele im Sketch
Bibliotheken	Schnittstellen und Grundfunktionen	DCF77, RTClib, Wire, TimeLib, ezButton
Zustand	Verfügbarkeit, Gültigkeit und Zeitbezug	rtcIsPresent, rtcTimeIsValid, dcfEverSynchronized
Initialisierung	Pins, Selbsttest und Startwerte	setup(), blankNixieDisplay(), dcfReceiver.Start()
Zyklischer Betrieb	Ereignisse prüfen und Ausgaben aktualisieren	loop(), Taster, RTC-Abgleich, Status-LED
Anzeigeabstraktion	Zeitwert in Registerbytes umsetzen	writeNixieDisplay(), shiftNixieBytes()

3 Zeitmodell und Priorität der Zeitquellen

3.1 TimeLib als aktive Systemzeit

Die im Programm verwendete laufende Uhr ist die Systemzeit der Time-Library. now() liefert einen time_t-Zeitwert; setTime() setzt diesen Wert entweder aus einem vollständigen Zeitstempel oder aus einzelnen Kalenderfeldern. Die Anzeige liest nicht unmittelbar DCF77 oder den DS3231, sondern immer einen zuvor erfassten TimeLib-Zeitwert.

Dadurch bleibt der Programmablauf auch dann funktionsfähig, wenn vorübergehend weder DCF77 noch RTC verfügbar sind. Die interne Zeit läuft zwischen zwei Korrekturen selbstständig weiter. Ihre Langzeitgenauigkeit hängt ohne externe Referenz jedoch vom Mikrocontroller-Takt und von der Implementierung der Time-Library ab.

3.2 Rolle des DS3231

Beim Start werden zwei voneinander unabhängige Eigenschaften ermittelt. rtcIsPresent gibt an, ob das Modul über I2C erreichbar war. rtcTimeIsValid beschreibt dagegen, ob die RTC laut Lost-Power-Flag eine verwendbare Zeit enthält. Diese Trennung ist wichtig: Eine elektrisch erreichbare RTC kann nach leerer oder fehlender Pufferbatterie einen ungültigen Zeitinhalt besitzen.

Aktueller RTC-Starttest

```
rtcIsPresent = rtcModule.begin();
if (rtcIsPresent) {
  rtcTimeIsValid = !rtcModule.lostPower();
}
```

Nur wenn beide Bedingungen erfüllt sind, wird der DS3231 einmal bei jedem neuen Systemsekundenwert gelesen. Dabei übernimmt setTime() Stunde, Minute, Sekunde, Tag, Monat und Jahr. Das vollständige Datum ist auch dann sinnvoll, wenn die Anzeige nur HH:MM zeigt: Es verhindert eine inkonsistente interne Kalenderzeit am Tageswechsel.

3.3 Rolle des DCF77-Empfangs

Die DCF77-Bibliothek verarbeitet das externe Pulssignal. Der Sketch fragt in jedem ungefähr 100 ms langen Schleifendurchlauf getTime() ab. Solange kein neues, von der Bibliothek akzeptiertes Telegramm bereitsteht, ist der Rückgabewert null. Ein Wert ungleich null wird als vollständiger Zeitstempel behandelt.

Eine gültige DCF-Zeit setzt sofort die TimeLib-Systemzeit. Ist der DS3231 vorhanden, wird derselbe Zeitstempel zusätzlich in die RTC geschrieben und deren Zeit anschließend als gültig markiert. Dadurch kann DCF77 eine zuvor wegen Lost Power ungültige, aber erreichbare RTC wieder in Betrieb nehmen.

3.4 DCF-Aktualität mit millis()

Die Statusanzeige benötigt keine zweite Uhrzeit, sondern nur die Information, wie lange die letzte DCF-Korrektur zurückliegt. Dafür speichert lastDcfSyncMillis den monoton fortlaufenden millis()-Zähler. Manuelle Zeitänderungen oder ein Sprung der RTC können diesen Alterswert deshalb nicht verfälschen.

Bewertung einer aktuellen DCF-Synchronisation

```
const bool dcfIsRecent =
    dcfEverSynchronized &&
    (currentMillis - lastDcfSyncMillis < DCF_SYNC_TIMEOUT_MS);
```

DCF_SYNC_TIMEOUT_MS beträgt 120 000 ms. Die Subtraktion zweier unsigned-long-Werte ist auch beim Überlauf des millis()-Zählers korrekt, solange das betrachtete Zeitintervall deutlich kleiner als die gesamte Zählerperiode ist. Das zusätzliche Flag dcfEverSynchronized verhindert, dass der Startwert null irrtümlich als frische Synchronisation gilt.

3.5 Praktische Priorität

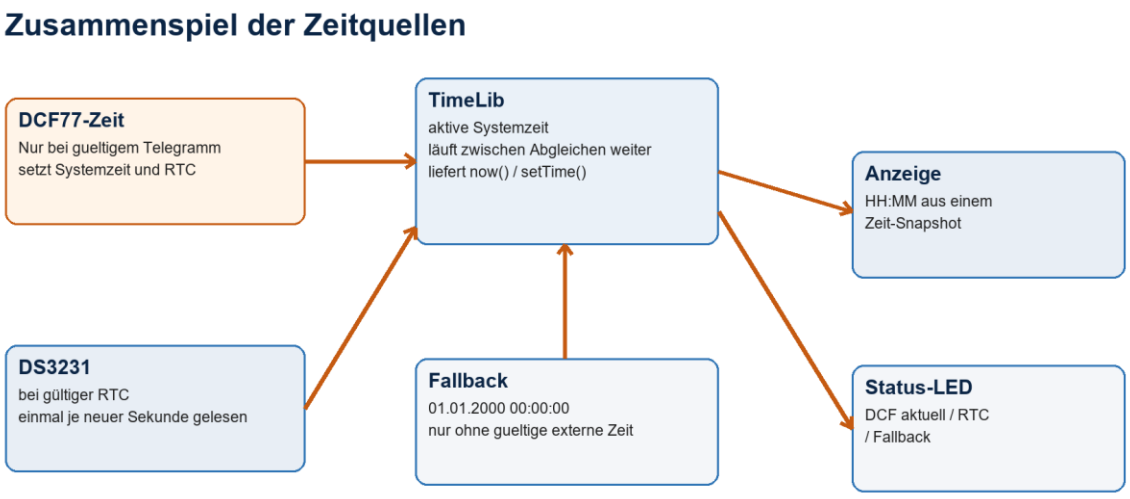


Abbildung 2: Zeitquellen und ihre Wirkung auf die Systemzeit

Die Priorität entsteht aus den ausgeführten Aktionen: Ein neues DCF-Telegramm ist die höchste absolute Korrektur und aktualisiert zugleich die RTC. Im Normalbetrieb wird danach der DS3231 einmal pro Sekunde als laufende Referenz übernommen. Fehlen gültige externe Daten, bleibt ausschließlich die TimeLib-Systemzeit. Die LED-Zustände beschreiben diese Vertrauenslage; sie sind keine formale Zustandsmaschine.

Wichtige Unterscheidung: Die Anzeige 'DCF aktuell' bedeutet, dass kürzlich eine DCF-Korrektur empfangen wurde. Sie bedeutet nicht, dass DCF77 kontinuierlich den Sekundentakt liefert. Zwischen Telegrammen läuft die Uhr über TimeLib und gegebenenfalls den DS3231 weiter.

4 Hardwarebezug, Pins und Bibliotheken

4.1 Mikrocontrolleranschlüsse

Symbol im Sketch	Arduino-Pin	ATmega-Funktion	Verwendung
PIN_DCF_INPUT	D2	PD2 / INT0	Aktiv-niedriges DCF77-Signal
PIN_BUTTON_MINUTE	D3	PD3	Minutentaster, externe Beschaltung
PIN_BUTTON_HOUR	D4	PD4	Stundentaster, externe Beschaltung
PIN_DRV_CLK	D5	PD5	Schiebetakt für beide CD4094
PIN_DRV_STR	D6	PD6	Strobe/Übernahme der Ausgangsspeicher
PIN_DRV_DAT	D7	PD7	Serielle Anzeigedaten
PIN_STATUS_LED	D13	PB5 / SCK	Status-LED; bei ISP auch Taktleitung
Wire SDA	A4	PC4 / SDA	I2C-Datenleitung zum DS3231
Wire SCL	A5	PC5 / SCL	I2C-Taktleitung zum DS3231

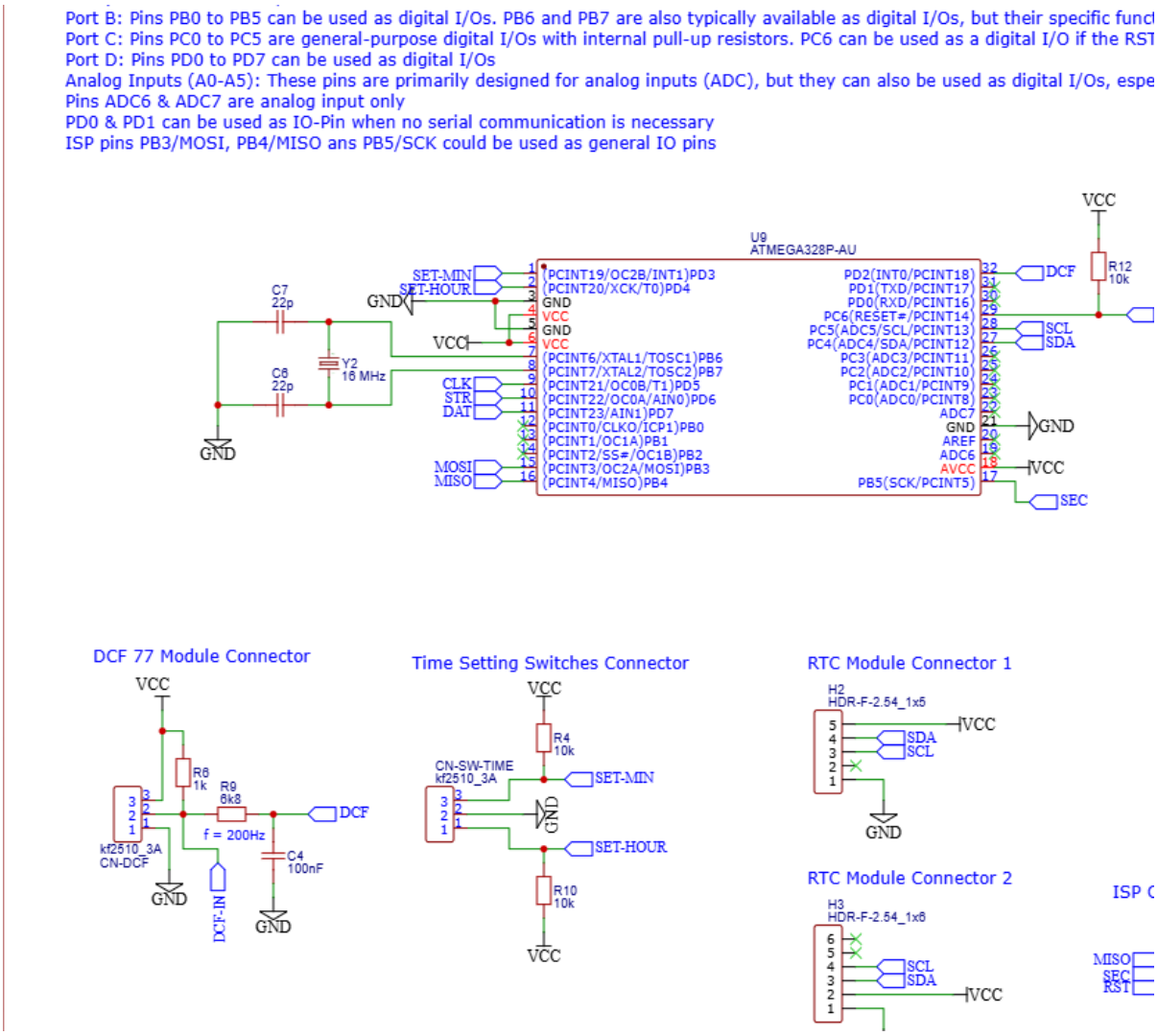


Abbildung 3: Mikrocontroller, DCF77-, Taster- und RTC-Anschlüsse im Schaltbild

Die I2C-Pins werden nicht als eigene Konstanten geführt, weil Wire auf dem ATmega328P automatisch die Hardwareanschlüsse A4 und A5 verwendet. Das vermeidet Definitionen, die zwar dokumentarisch aussehen, aber keinen Einfluss auf die Bibliothek hätten.

4.2 Externe Eingangsbeschaltung

Der DCF77-Eingang ist auf der Leiterplatte als Open-Collector-Signal mit Pull-up und RC-Entstörung ausgeführt. Im DCF77-Objekt wählt der dritte Konstruktorparameter false die auf dieser Hardware getestete aktiv-niedrige Polarität. Der externe Interrupt 0 ist beim ATmega328P fest mit D2 beziehungsweise PD2 verbunden.

Auch die Einstelltaster besitzen externe Pull-up-Widerstände und ziehen den jeweiligen Eingang beim Betätigen gegen Masse. Daher verwendet der Sketch INPUT und nicht INPUT_PULLUP. Die gewählte

ezButton-Konfiguration ist eine hardwarebezogene Besonderheit: isReleased() entspricht in dieser Kombination derjenigen logischen Flanke, die beim realen Drücken des Tasters entsteht.

Nicht intuitiver Methodenname: Die Bedienfunktion wird physisch beim Drücken ausgelöst, obwohl der Quelltext isReleased() abfragt. Diese Zuordnung wurde an der vorhandenen Schaltung getestet und darf nicht allein nach dem Methodennamen umgedreht werden.

4.3 Eingebundene Bibliotheken

Bibliothek	Aufgabe im Programm	Relevante Schnittstellen
DCF77.h	Decodierung des DCF77-Pulssignals	Start(), getTime()
Time.h / TimeLib	Laufende Systemzeit und Kalenderzerlegung	now(), setTime(), hour(), minute(), second(), day(), month(), year()
RTClib	Zugriff auf den DS3231	begin(), lostPower(), now(), adjust(), DateTime
Wire	I2C-Kommunikation	wird über RTClib verwendet
ezButton	Tasterabfrage und Entprellung	loop(), isReleased()
Arduino-Core	GPIO, Zeitverzögerung und Bitübertragung	pinMode(), digitalWrite(), shiftOut(), delay(), millis()

Das genaue interne Verhalten der Bibliotheken ist versionsabhängig. Für eine reproduzierbare Veröffentlichung sollten deshalb die verwendeten Bibliotheksversionen zusammen mit dem Sketch dokumentiert oder in einer Projektdatei festgehalten werden.

5 Programmstart und Selbsttest

5.1 Reihenfolge in setup()

1. Die serielle Schnittstelle wird nur bei ENABLE_SERIAL_DEBUG ungleich null mit 57 600 Baud initialisiert.
2. Der DS3231 wird gesucht; Erreichbarkeit und Zeitgültigkeit werden getrennt gespeichert.
3. Status-LED, DCF77-Eingang und die drei CD4094-Steuerleitungen werden konfiguriert.
4. CLK, STR und DAT erhalten definierte LOW-Pegel. Anschließend werden die Nixie-Ausgänge sofort dunkel geschaltet.
5. Die beiden Tasterpins werden mit der hardwaregeprüften INPUT-Konfiguration eingerichtet.
6. Die Status-LED wird 30-mal mit jeweils 80 ms Wartezeit umgeschaltet.

7. Die Stundenanzeige durchläuft 00:00 bis 23:00; danach läuft die Minutenanzeige bei 23:xx von 00 bis 59.
8. Eine definierte Fallback-Zeit 01.01.2000, 00:00:00 wird gesetzt und kurz angezeigt.
9. Erst nach dem vollständig blockierenden Selbsttest startet der DCF77-Decoder.
10. lastDisplayedSecond wird auf die aktuelle Systemsekunde gesetzt, um keine doppelte Sofortaktualisierung auszulösen.

5.2 Definierte Nixie-Ausgänge

Unmittelbar nach dem Einschalten können Schieberegister undefinierte Inhalte enthalten. Würde der Selbsttest vor der ersten gezielten Ausgabe beginnen, könnten zufällige Ziffern oder mehrere Kathoden sichtbar werden. Der Sketch setzt daher zuerst die Steuerleitungen und ruft blankNixieDisplay() auf, noch bevor längere delay()-Phasen beginnen.

Die Dunkelcodes nutzen nicht angeschlossene beziehungsweise ungültige Dekoderkombinationen. Dadurch bleibt die Anzeige in einem bekannten dunklen Zustand, ohne dass zusätzliche Blank-Eingänge erforderlich sind.

5.3 Selbsttest ohne Veränderung der Uhrzeit

Der Selbsttest ruft writeNixieDisplay(h, m) mit expliziten Zahlenwerten auf. Diese Funktion erzeugt nur Registerbytes und greift nicht auf die globale TimeLib-Uhr zu. Damit kann der gesamte Ziffernbereich geprüft werden, ohne die spätere Systemzeit fortlaufend auf Testwerte zu setzen.

Die sichtbare Startphase dauert näherungsweise 2,4 s für die LED, 2,4 s für die Stunden, 6,0 s für die Minuten und 1,5 s für die Fallback-Anzeige, insgesamt also etwa 12,3 s zuzüglich Programmlaufzeit. DCF77 wird erst danach gestartet. Diese Verzögerung ist bewusst akzeptiert, weil der Starttest immer aktiv bleiben soll.

6 Erzeugung der Nixie-Anzeigedaten

6.1 Von HH:MM zu vier Dezimalziffern

writeNixieDisplay() erhält Stunde und Minute als getrennte Ganzzahlen. Division durch 10 liefert die Zehnerstelle; der Restoperator % 10 liefert die Einerstelle. Jede Dezimalziffer wird anschließend als BCD-Wert in die zur Verdrahtung gehörenden Bitpositionen eingesetzt.

Eine vollständige Ziffer von 0 bis 9 benötigt vier BCD-Bits A bis D, wobei A das niederwertigste Bit ist. Wegen der begrenzten Wertebereiche genügen für den Stundenzehner zwei Bits und für den Minutenzehner drei Bits. Insgesamt werden daher nur 13 der 16 CD4094-Ausgänge für gültige Zifferninformationen benötigt.

6.2 Belegung der beiden Bytes

Byte	Bits	Bedeutung	Wertebereich
hourByte	1..0	Stunden-Zehner, BCD A/B	0 bis 2; Code 3 zum Ausblenden
hourByte	5..2	Stunden-Einer, BCD A/B/C/D	0 bis 9
hourByte	7..6	nicht verwendet	0
minuteByte	2..0	Minuten-Zehner, BCD A/B/C	0 bis 5
minuteByte	6..3	Minuten-Einer, BCD A/B/C/D	0 bis 9
minuteByte	7	nicht verwendet	0

Prinzip der Bitbildung

```
hourByte |= ((displayHour / 10) & 0x03);
hourByte |= ((displayHour % 10) & 0x0F) << 2;
minuteByte |= ((displayMinute / 10) & 0x07);
minuteByte |= ((displayMinute % 10) & 0x0F) << 3;
```

6.3 Übertragungsreihenfolge und Registerkaskade

Der CD4094, der die Minuten ausgibt, liegt seriell näher am Mikrocontroller. Sein serieller Ausgang speist das nachgeschaltete Stundenregister. Deshalb muss das Stundenbyte zuerst gesendet werden: Während anschließend das Minutenbyte folgt, wird das zuerst gesendete Byte durch das erste Register hindurch in das zweite weitergeschoben.

shiftOut(..., MSBFIRST, ...) sendet innerhalb jedes Bytes Bit 7 zuerst und Bit 0 zuletzt. Nach 16 Taktflanken befinden sich die acht zuletzt gesendeten Minutenbits im ersten Register und die zuvor gesendeten Stundenbits im nachgeschalteten Register.

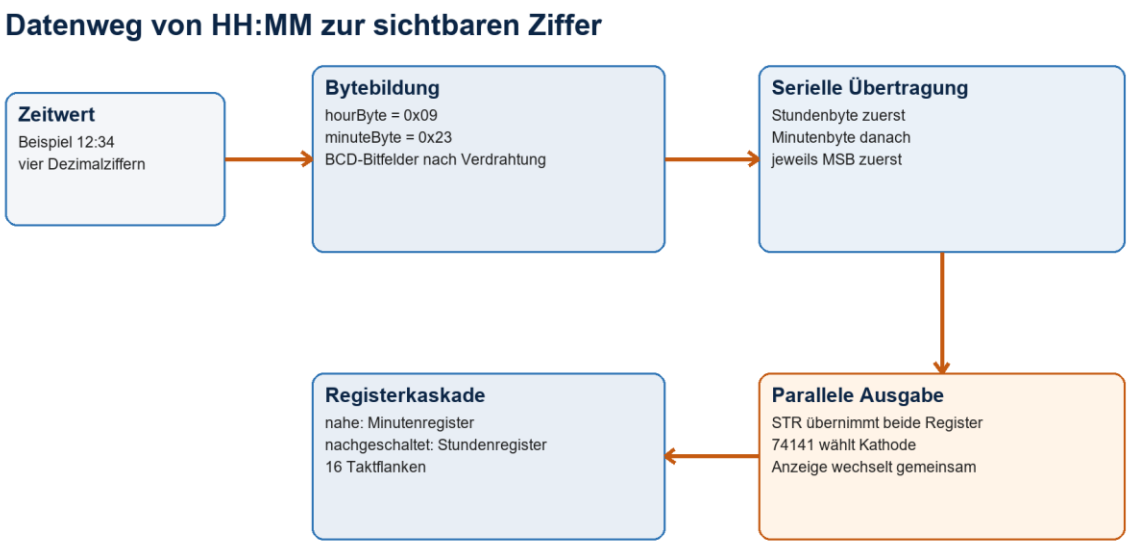


Abbildung 4: Bildung, Übertragung und parallele Ausgabe der Anzeigedaten

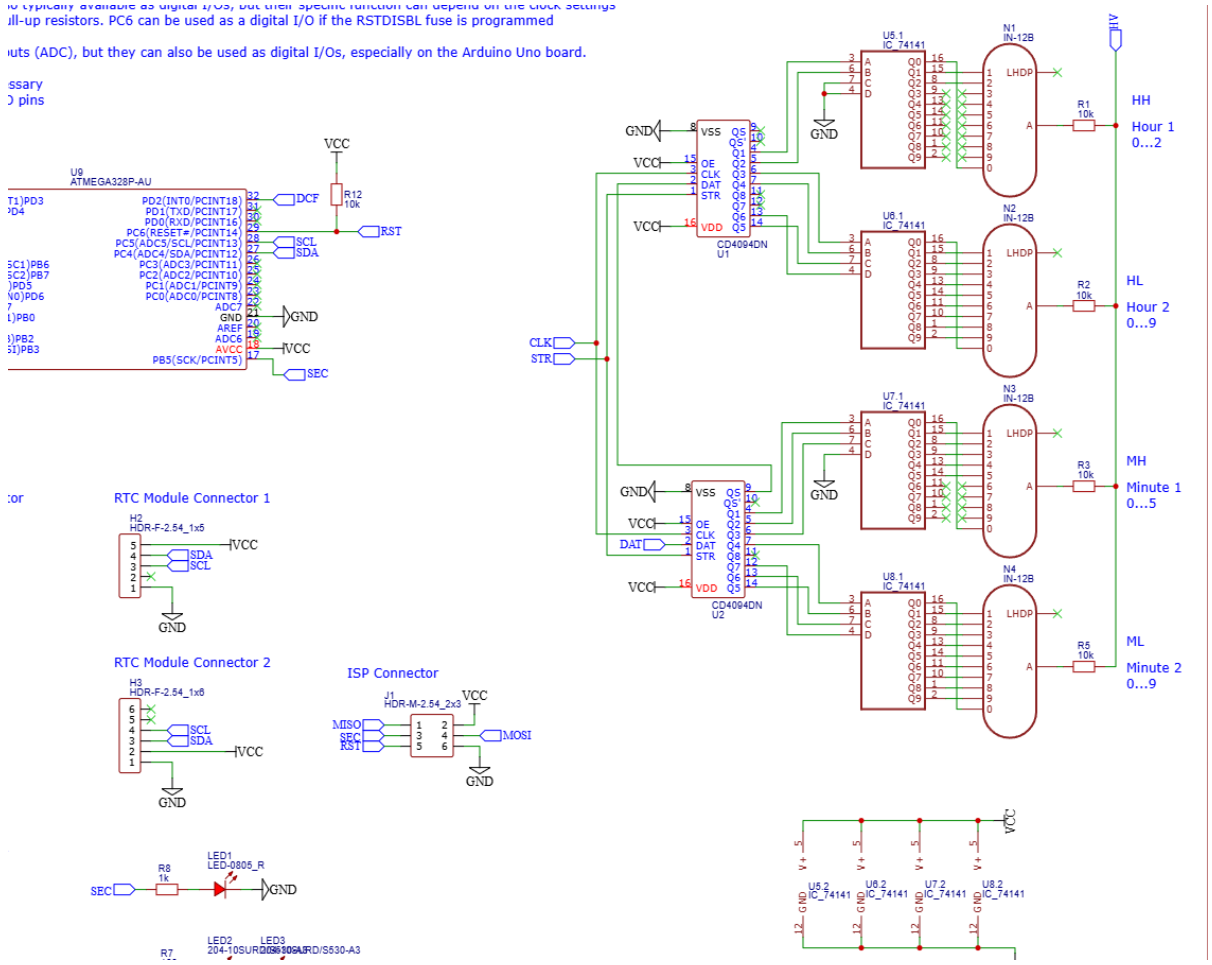


Abbildung 5: CD4094-, 74141- und Nixie-Stufe im vorhandenen Schaltbild

6.4 Oszilloskopaufnahme an der realen Hardware

Die drei Eingangssignale des unteren, dem Mikrocontroller näheren CD4094 wurden während einer Anzeigeaktualisierung mit dem Oszilloskop aufgenommen. Kanal C1 zeigt CLOCK, Kanal C2 DATA und Kanal C3 STROBE. Die Zeitbasis beträgt 50 µs pro Division; für CLOCK wird eine Frequenz von rund 61,24 kHz angezeigt.

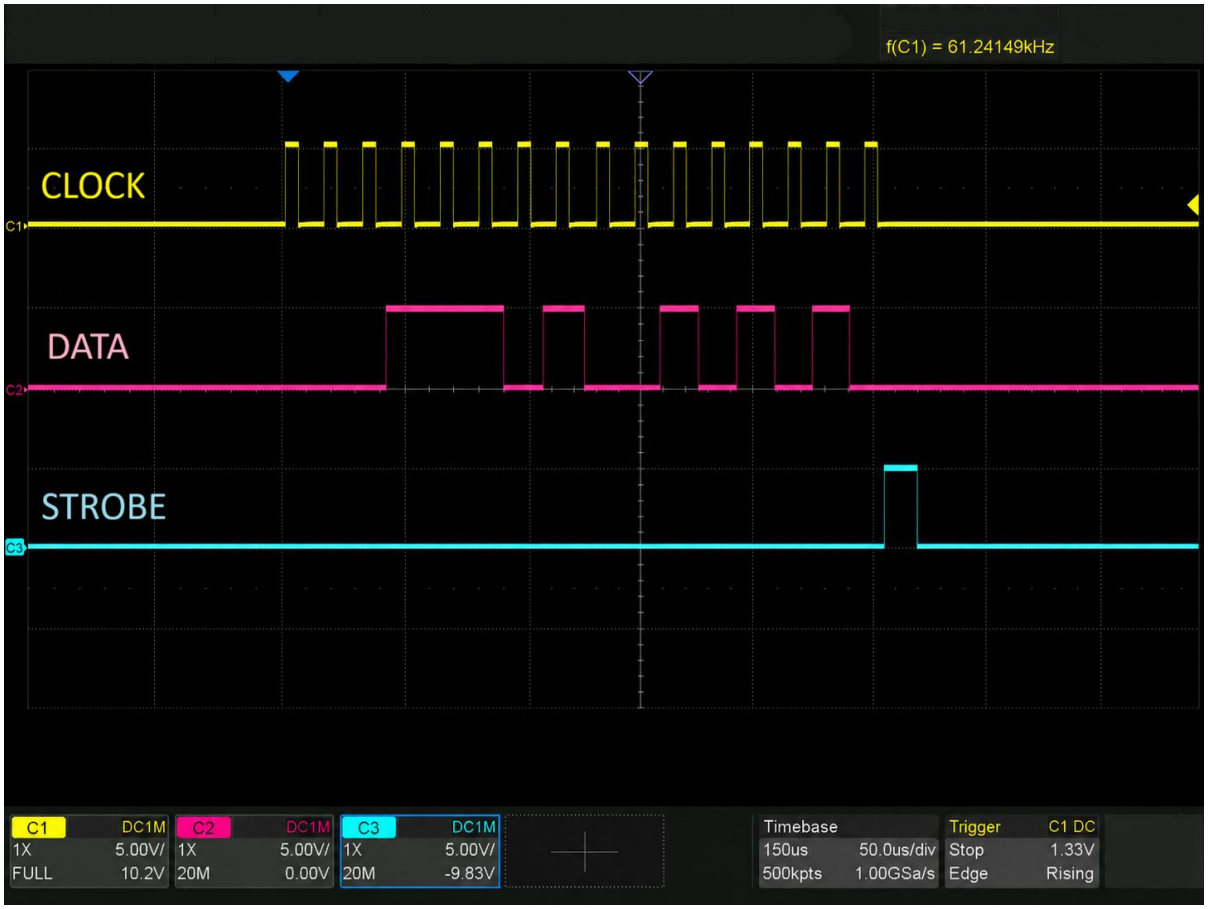


Abbildung 6: CLOCK, DATA und STROBE am Eingang des unteren CD4094

Signal	Beobachtung	Bezug zum Programm
CLOCK / C1	Genau 16 positive Taktimpulse mit rund 61,24 kHz	Zwei shiftOut()-Aufrufe zu je acht Bit
DATA / C2	00011101 00101010 an den positiven Taktflanken	0x1D wird zuerst, 0x2A danach übertragen
STROBE / C3	Ein positiver Impuls nach dem 16. Takt	Gemeinsame Übernahme beider Ausgangsspeicher

Das erste Byte 0x1D ist das Stundenbyte: Bits 1..0 ergeben den Stundenzehner 1, Bits 5..2 den Stundeneiner 7. Das zweite Byte 0x2A ist das Minutenbyte: Bits 2..0 ergeben den Minutenzehner 2, Bits 6..3 den Minuteneiner 5. Die Aufnahme dokumentiert damit die Übertragung für die Anzeige 17:52.

DATA liegt jeweils vor der positiven CLOCK-Flanke stabil an. STROBE bleibt während der gesamten seriellen Übertragung LOW und wird erst nach dem letzten Takt kurz HIGH. Die Messung bestätigt damit sowohl die Byte-Reihenfolge als auch die flackerfreie gemeinsame Übernahme der parallelen Ausgänge.

6.5 Strobe und flackerfreie Übernahme

Während des Einschiebens bleibt STR LOW. Die internen Schieberegister verändern sich bei jedem CLK-Impuls, die parallelen Ausgangsspeicher behalten jedoch den bisherigen Anzeigewert. Erst nach beiden shiftOut()-Aufrufen erzeugt das Programm einen HIGH-LOW-Impuls an STR. Dadurch werden beide Ausgangsspeicher gemeinsam aktualisiert.

Serielle Übertragung und gemeinsame Übernahme

```
digitalWrite(PIN_DRV_STR, LOW);
shiftOut(PIN_DRV_DAT, PIN_DRV_CLK, MSBFIRST, hourByte);
shiftOut(PIN_DRV_DAT, PIN_DRV_CLK, MSBFIRST, minuteByte);
digitalWrite(PIN_DRV_STR, HIGH);
digitalWrite(PIN_DRV_STR, LOW);
```

Ohne den getrennten Ausgangsspeicher würden die während der 16 Taktflanken entstehenden Zwischenmuster kurz an den 74141 erscheinen. Die Kombination aus Schieberegister und Latch verhindert diese sichtbaren Mischzustände.

6.6 Führende Null und vollständige Dunkeltastung

Bei Stunden unter 10 soll die linke Röhre dunkel bleiben. Der Stundenzehner besitzt nur die angeschlossenen Auswahlmöglichkeiten 0 bis 2. Der Code 3 setzt A und B auf HIGH und wählt damit einen nicht angeschlossenen Dekoderausgang. Die Einerstellen verwenden für vollständige Dunkeltastung den ungültigen BCD-Code 15.

Funktion	hourByte	minuteByte	Wirkung
Normale Anzeige	abhängig von Stunde	abhängig von Minute	Vier Ziffern beziehungsweise ausgeblendete führende Null
blankNixieDisplay()	0x3F	0x7F	Stundenzehner 3, Minutenzehner 7, beide Einerstellen 15

6.7 Rechenbeispiel 12:34

Für 12:34 entstehen vier Ziffern: 1, 2, 3 und 4. Der Stundenzehner 1 belegt Bits 1..0 mit 01. Der Stundeneiner 2 wird um zwei Positionen nach links verschoben und ergibt 001000. Zusammen entsteht hourByte = 00001001b = 0x09.

Beim Minutenbyte liefert der Minutenzehner 3 die unteren Bits 011. Der Minuteneiner 4 wird um drei Positionen verschoben und ergibt 0100000. Zusammen entsteht minuteByte = 00100011b = 0x23. Gesendet werden zuerst 0x09 und danach 0x23, jeweils mit dem höchstwertigen Bit zuerst.

Teil	Dezimalwert	Position	Beitrag zum Byte
Stunden-Zehner	1	hourByte Bits 1..0	00000001
Stunden-Einer	2	hourByte Bits 5..2	00001000
Ergebnis Stunde	12	hourByte	00001001 = 0x09
Minuten-Zehner	3	minuteByte Bits 2..0	00000011
Minuten-Einer	4	minuteByte Bits 6..3	00100000
Ergebnis Minute	34	minuteByte	00100011 = 0x23

7 Ablauf der Hauptschleife

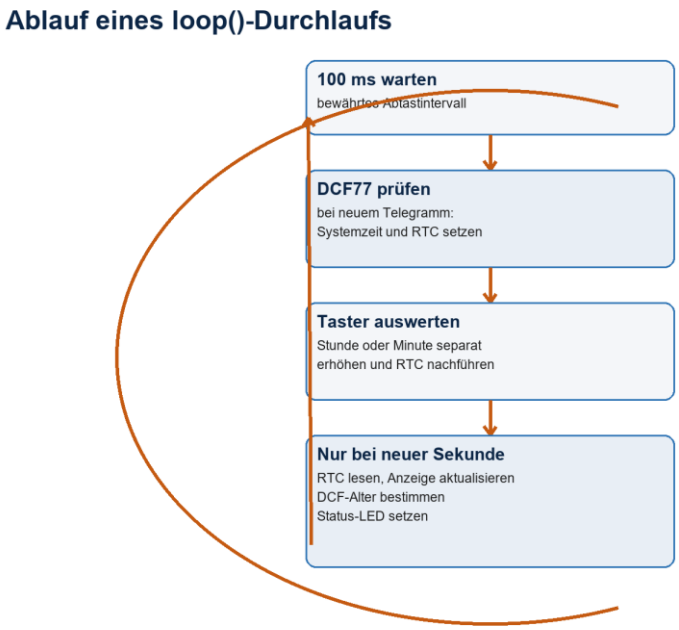


Abbildung 7: Reihenfolge der Aufgaben in loop()

7.1 Grundrhythmus

Jeder Durchlauf beginnt mit delay(100). Dadurch werden DCF-Rückgabewert und Taster ungefähr zehnmal pro Sekunde geprüft. Dieses Intervall ist Teil der auf der vorhandenen Hardware bewährten Tasterlogik. Die Anzeige und der RTC-Abgleich sind davon entkoppelt und werden nur bei einem neuen Sekundenwert ausgeführt.

7.2 Verarbeitung einer neuen DCF-Zeit

1. getTime() wird abgefragt.
2. Bei einem Rückgabewert ungleich null setzt setTime(dcfTime) die aktive Systemzeit.
3. millis() wird als Zeitpunkt der letzten DCF-Korrektur gespeichert und dcfEverSynchronized wird true.
4. Eine vorhandene RTC erhält denselben Zeitstempel; rtcTimeIsValid wird anschließend true.

Die RTC wird in diesem Fall unabhängig von ihrem bisherigen Lost-Power-Zustand geschrieben. Nur ihre physische Erreichbarkeit beim Start ist Voraussetzung.

7.3 Verarbeitung der Taster

buttonHour.loop() und buttonMinute.loop() müssen regelmäßig aufgerufen werden, damit ezButton den Eingangszustand und die Entprellung nachführen kann. Bei einer erkannten physischen Betätigung wird genau ein Feld geändert.

Taster	Änderung	Überlauf	Unveränderte Felder
Stunde	+1 Stunde	23 wird 00	Minute, Sekunde und Datum
Minute	+1 Minute; Sekunde wird 0	59 wird 00, ohne Stundenerhöhung	Stunde und Datum

Minuteneinstellung ohne Übertrag auf die Stunde

```
const uint8_t newMinute = (minute(buttonTime) + 1) % 60;
setTime(hour(buttonTime), newMinute, 0,
        day(buttonTime), month(buttonTime), year(buttonTime));
```

Vor jeder Berechnung wird now() genau einmal in buttonTime gespeichert. Stunde, Minute, Sekunde und Datum stammen damit aus derselben Momentaufnahme. Nach der Änderung wird eine vorhandene RTC neu gestellt. Eine manuelle Einstellung kann deshalb auch eine erreichbare RTC mit vorher ungültiger Zeit wieder als gültig markieren.

7.4 Einmalige Aktualisierung je Sekunde

Nach DCF- und Tasterverarbeitung wird currentTime = now() erfasst. Nur wenn dieser Wert von lastDisplayedSecond abweicht, folgen RTC-Abgleich, Anzeige und LED. Damit sinkt die I2C-Belastung gegenüber einer Abfrage in jedem 100-ms-Durchlauf auf ungefähr einen Zugriff pro Sekunde.

Ist die RTC vorhanden und als gültig markiert, prüft lostPower() erneut den Zustand. Bei gesetztem Flag wird rtcTimeIsValid false. Andernfalls liest rtcModule.now() alle Kalenderfelder; setTime() übernimmt Datum und Uhrzeit vollständig. Danach wird currentTime erneut erfasst, damit Anzeige, gespeicherte Sekunde und LED denselben korrigierten Zeitstand verwenden.

7.5 Konsistente Zeit-Snapshots

Mehrere direkte now()-Aufrufe innerhalb einer Berechnung könnten theoretisch verschiedene Sekunden liefern. Der Sketch erfasst deshalb für jeden zusammengehörigen Ablauf einen lokalen time_t-Wert: buttonTime für eine Tasteraktion, currentTime für RTC-, Anzeige- und LED-Aktualisierung und adjustedTime für das Schreiben einer manuell korrigierten Zeit in die RTC.

Begriffspräzision: Die lokale Momentaufnahme macht die verwendeten Zeitfelder logisch konsistent. Sie ist nicht mit einer allgemeinen Interruptsperre oder einer atomaren Hardwaretransaktion gleichzusetzen.

8 Bedienung und Statusanzeige

8.1 Bedienphilosophie

Jede erkannte Tasterbetätigung löst genau einen Schritt aus. Es gibt keine automatische Wiederholfunktion und keinen natürlichen Übertrag in die nächsthöhere Einheit. Diese Bedienung eignet sich für eine Uhr, bei der Stunde und Minute unabhängig korrigiert werden sollen.

Der Stundentaster behält Minute und Sekunde bei. Der Minutentaster setzt die Sekunde bewusst auf null, damit eine Einstellung auf einen beobachteten Minutenwechsel möglich ist. Bei 59 Minuten springt nur die Minute auf 00; die Stunde bleibt unverändert. Analog springt die Stunde bei 23 auf 00, ohne das Datum zu erhöhen.

8.2 Bedeutung der Status-LED

LED-Zustand	Bedingung	Interpretation
Blinkend im Sekundentakt	DCF-Telegramm vor weniger als 120 s empfangen	Uhr wurde kürzlich per Funkzeit korrigiert

LED-Zustand	Bedingung	Interpretation
Aus	DCF nicht aktuell, RTC-Zeit gültig	DS3231 steht als verlässliche lokale Referenz zur Verfügung
Dauerhaft ein	weder aktuelle DCF-Korrektur noch gültige RTC	Uhr läuft nur mit der internen TimeLib-Zeit

Das Blinksignal verwendet second(currentTime) % 2. Die LED wechselt damit bei jeder Sekunde zwischen LOW und HIGH. Die Statusentscheidung wird ebenfalls nur einmal pro neuer Sekunde getroffen.

9 Funktionen und globale Zustände im Detail

9.1 Funktionsreferenz

Funktion	Eingaben	Wirkung und Seiteneffekte
shiftNixieBytes	hourByte, minuteByte	Sendet 16 Bits und strobt beide CD4094; verändert die sichtbare Anzeige.
writeNixieDisplay	displayHour, displayMinute	Bildet BCD-Bitfelder; verändert keine Uhrzeit; ruft shiftNixieBytes auf.
updateNixieDisplay	displayTime	Extrahiert Stunde und Minute aus einem Snapshot und zeigt sie an.
blankNixieDisplay	keine	Gibt 0x3F und 0x7F aus und schaltet alle Röhren dunkel.
setup	keine	Initialisiert Module und Pins, führt Selbsttest aus und startet DCF77.
loop	keine	Verarbeitet DCF, Taster, RTC, Anzeige und Status-LED zyklisch.

9.2 Globale Zustandsvariablen

Variable	Typ	Bedeutung	Änderung
lastDisplayedSecond	time_t	zuletzt vollständig verarbeiteter Sekundenwert	am Ende der 1-Hz-Aktualisierung
lastDcfSyncMillis	unsigned long	monotoner Zeitpunkt der letzten DCF-Korrektur	bei gültigem DCF-Rückgabewert
dcfEverSynchronized	bool	mindestens eine DCF-Zeit wurde empfangen	false beim Start, danach true

Variable	Typ	Bedeutung	Änderung
rtcIsPresent	bool	DS3231 war beim Start erreichbar	einmal in setup()
rtcTimeIsValid	bool	RTC enthält nach Programmlogik verwendbare Zeit	Startprüfung, Lost Power, DCF oder manuelle Einstellung

9.3 Konstanten und Objekte

Die Pin- und Zeitwerte sind als constexpr mit passenden Datentypen angelegt. Dadurch sind sie unveränderlich und benötigen keine unnötig breiten int-Variablen. DCF_INTERRUPT bleibt als Interruptnummer 0 getrennt vom Pin D2, weil die verwendete Bibliothek beide Angaben erwartet.

rtcModule kapselt den DS3231. dcfReceiver verbindet Pin D2, Interrupt 0 und die aktiv-niedrige Signalpolarität. buttonHour und buttonMinute kapseln die Taster mit der exakt hardwaregetesteten INPUT-Konfiguration.

10 Typische Betriebssituationen und Grenzfälle

10.1 Start mit gültiger RTC

rtcIsPresent und rtcTimeIsValid werden true. Nach dem immer aktiven Selbsttest zeigt das Programm zunächst kurz die Fallback-Zeit. Beim nächsten erkannten Sekundenwechsel wird der DS3231 gelesen, die TimeLib vollständig korrigiert und die RTC-Zeit angezeigt. Ohne aktuelle DCF-Korrektur bleibt die Status-LED aus.

10.2 RTC vorhanden, aber Lost Power

rtcIsPresent wird true, rtcTimeIsValid bleibt false. Die Uhr läuft zunächst ab der Fallback-Zeit und die LED leuchtet dauerhaft. Ein gültiges DCF-Telegramm oder eine manuelle Einstellung schreibt die RTC und setzt rtcTimeIsValid auf true. Danach kann der DS3231 wieder als Zeitquelle dienen.

10.3 Keine RTC und kein DCF77

Die Uhr bleibt funktionsfähig, besitzt aber nur die interne TimeLib-Zeit. Nach jedem Neustart beginnt sie bei 01.01.2000, 00:00:00. Die LED leuchtet dauerhaft. Tasteränderungen wirken auf die laufende Systemzeit, können mangels RTC jedoch nicht über einen Spannungsverlust hinweg gespeichert werden.

10.4 DCF77-Empfang nach längerer Unterbrechung

Nach 120 Sekunden ohne neue DCF-Korrektur endet der Blinkstatus. Bei gültiger RTC geht die LED aus; ohne gültige RTC bleibt sie an. Ein späteres gültiges Telegramm setzt Systemzeit und RTC erneut und startet das 120-s-Zeitfenster neu.

10.5 Manuelle Grenzwerte

Ausgangswert	Betätigung	Ergebnis	Bewusste Eigenschaft
23:17	Stunde	00:17	kein Datumsübertrag
08:59	Minute	08:00	kein Stundenübertrag; Sekunde = 0
23:59	Minute	23:00	Stunde bleibt 23
23:59	erst Minute, dann Stunde	00:00	beide Felder werden getrennt eingestellt

10.6 millis()-Überlauf

Auf einem typischen Arduino mit 32-Bit unsigned long läuft millis() nach rund 49,7 Tagen über. Die Differenz currentMillis - lastDcfSyncMillis bleibt aufgrund der modularen unsigned-Arithmetik für das kurze 120-s-Fenster korrekt. Es ist deshalb kein Sonderfall beim Überlauf erforderlich.

11 Übersetzung, Diagnose und Inbetriebnahme

11.1 Zielplattform

Der geprüfte Build verwendet ein Arduino-Nano-kompatibles Board mit ATmega328P bei 16 MHz und die Prozessorvariante 'ATmega328P (Old Bootloader)'. Die endgültige Hardware kann als Minimalbeschaltung aufgebaut sein; für die Programmlogik entscheidend bleiben dieselbe Taktfrequenz und die dokumentierte Pinbelegung.

Konfiguration	Programmspeicher	Dynamischer Speicher
ENABLE_SERIAL_DEBUG = 0	10 498 Byte (34 %)	401 Byte (19 %)
ENABLE_SERIAL_DEBUG = 1	11 480 Byte (37 %)	576 Byte (28 %)

Die Werte stammen aus der erfolgreichen Kompilierung des dokumentierten Programmstands für arduino:avr:nano:cpu=atmega328old. Andere Core- oder Bibliotheksversionen können geringfügig andere Größen erzeugen.

11.2 Serieller Compilerschalter

Compile-Time-Schalter für die serielle Schnittstelle

```
#ifndef ENABLE_SERIAL_DEBUG
#define ENABLE_SERIAL_DEBUG 0
#endif

#if ENABLE_SERIAL_DEBUG
  Serial.begin(57600);
#endif
```

Die Veröffentlichungsvoreinstellung ist 0. Der Wert kann im Sketch oder über eine Compilerdefinition wie -DENABLE_SERIAL_DEBUG=1 geändert werden. Im aktuellen Programm wird dadurch ausschließlich Serial.begin() eingebunden; es existieren noch keine Serial.print()-Ausgaben. Der Schalter stellt daher die technische Voraussetzung für spätere Diagnosemeldungen bereit, erzeugt gegenwärtig aber kein Protokoll.

11.3 Inbetriebnahmereihenfolge

1. Sketch für die korrekte Zielplatine und Bootloader-Variante kompilieren.
2. Zunächst ohne angeschlossene Hochspannung prüfen, ob Reset, Taster, I2C und DCF-Eingang plausible Pegel besitzen.
3. Den Starttest beobachten: LED-Umschaltung, Stundenlauf, Minutenlauf und definierte Fallback-Anzeige.
4. Mit gültiger RTC prüfen, ob nach dem Selbsttest die reale Zeit erscheint und die LED ohne DCF aus bleibt.
5. DCF77-Empfang prüfen; nach gültiger Synchronisation muss die LED im Sekundentakt blinken.
6. Taster an 23 Stunden und 59 Minuten auf das getrennte Modulo-Verhalten prüfen.

12 Systemgrenzen, Wartung und Erweiterbarkeit

12.1 Bewusst beibehaltene Eigenschaften

- Der Starttest ist immer aktiv und blockiert den DCF-Start für ungefähr 12,3 Sekunden.
- Die Tasterkonfiguration und die Verwendung von isReleased() entsprechen der vorhandenen, praktisch getesteten Hardware.
- Die Zeitquellenpriorität ist durch den Ablauf realisiert, nicht durch ein enum oder eine eigene Zustandsmaschine.
- Time.h und die vorhandene DCF77-Bibliothek bleiben Teil der bestehenden Entwicklungsbasis.

12.2 Aktuelle technische Grenzen

rtcIsPresent wird nur beim Programmstart bestimmt. Wird ein zunächst fehlendes RTC-Modul später angeschlossen oder erholt sich der I2C-Bus nach einer Störung, erfolgt keine automatische Neuerkennung. Ebenso prüft der Sketch beim laufenden rtcModule.now()-Aufruf keinen separaten Kommunikationsfehler.

Eine manuelle Einstellung schreibt auch dann ein Datum in die RTC, wenn die Systemzeit noch auf dem Fallback-Datum 01.01.2000 beruht. Für die HH:MM-Anzeige ist das unkritisch, für spätere Kalenderfunktionen müsste die Herkunft und Gültigkeit des Datums genauer modelliert werden.

Die Status-LED fasst mehrere Zustände bewusst grob zusammen. Sie zeigt weder die DCF-Signalqualität noch einzelne Telegrammfehler oder einen I2C-Kommunikationsfehler an. Für Diagnosezwecke wären zusätzliche serielle Meldungen oder ein expliziter Zeitquellenzustand mögliche Erweiterungen.

12.3 Regeln für spätere Änderungen

- Die Zuordnung der BCD-Bits nur gemeinsam mit Schaltbild und realer Anzeige ändern.
- Stundenbyte weiterhin vor dem Minutenbyte übertragen, solange die Registerkaskade unverändert bleibt.
- Tasteränderungen einzeln an der Hardware testen; die logische ezButton-Flanke nicht ohne Messung umstellen.
- Zeitfelder innerhalb eines Ablaufs aus einem gemeinsamen Snapshot ableiten.
- RTC-Erreichbarkeit, Zeitgültigkeit und DCF-Aktualität als getrennte Sachverhalte behandeln.
- Vor jeder Programmänderung eine eindeutig benannte .bck-Kopie der INO-Datei anlegen, damit die Arduino-IDE keine zweite Sketch-Datei kompiliert.

13 Prüfvorschläge für Hardware und Software

13.1 Funktionale Prüffälle

Prüffall	Vorgehen	Erwartetes Ergebnis
Definierter Start	Mehrfach ein- und ausschalten	Vor dem Ziffernlauf keine zufälligen sichtbaren Muster
Selbsttest	Kompletten Startablauf beobachten	24 Stundenwerte und 60 Minutenwerte ohne Veränderung der späteren Uhrzeit
RTC gültig	DCF abschirmen, RTC korrekt stellen	Zeit erscheint; LED bleibt aus
RTC Lost Power	Pufferzustand gezielt ungültig machen	Fallback und LED an; DCF kann RTC wieder aktivieren

Prüffall	Vorgehen	Erwartetes Ergebnis
DCF aktuell	Gültiges Telegramm abwarten	Zeitkorrektur, RTC-Korrektur und blinkende LED
DCF veraltet	Empfang nach Synchronisation unterbrechen	Nach 120 s Wechsel auf RTC- oder Fallback-Anzeige der LED
Stundentaster	Bei 23:xx einmal drücken	00:xx ohne Datumsänderung
Minutentaster	Bei xx:59 einmal drücken	xx:00, Sekunde 0, keine Stundenerhöhung
Langzeitlauf	Mehrere Tage betreiben	Keine sichtbaren Sprünge; millis()-Überlauf nach längerer Laufzeit ohne Statusfehler

13.2 Messpunkte bei Fehlersuche

Für die Anzeigeansteuerung sind DAT, CLK und STR die wichtigsten Messpunkte. Bei einer Aktualisierung müssen 16 CLK-Impulse sichtbar sein; DAT enthält die beiden Bytes, und STR erzeugt erst danach den Übernahmeimpuls. Bleibt die Anzeige unverändert, kann so zwischen fehlender Bytebildung, fehlendem Takt und fehlender Übernahme unterschieden werden.

Abbildung 6 zeigt einen fehlerfreien Referenzverlauf an der aufgebauten Uhr. Für einen Vergleich sind insbesondere die Anzahl der CLOCK-Impulse, die Stabilität von DATA an der positiven Taktflanke und die Lage des STROBE-Impulses nach dem vollständigen Datenblock maßgeblich.

Am DCF-Eingang ist die aktiv-niedrige Pulsfolge an D2 zu prüfen. Für die RTC sind SDA und SCL auf Busaktivität und korrekte Versorgung zu untersuchen. An den Tasterpins sollte der Ruhezustand HIGH und die Betätigung LOW ergeben. Wegen der gemeinsamen Verwendung von PB5/D13 als Status-LED und ISP-Takt ist während der Programmierung mit LED-Aktivität zu rechnen.

14 Begriffe und Dokumentationsbasis

14.1 Kurzglossar

Begriff	Bedeutung im Projekt
BCD	Binary Coded Decimal: Jede Dezimalziffer wird getrennt binär codiert.
CD4094	8-Bit-Schieberegister mit separatem parallelem Ausgangsspeicher.
74141	BCD-zu-Dezimal-Dekoder und Kathodentreiber für die Nixie-Röhre.
DCF77	Funkzeitsignal; die Bibliothek liefert nach erfolgreicher Decodierung einen Zeitstempel.
DS3231	Batteriegepufferte Echtzeituhr mit I2C-Schnittstelle.

Begriff	Bedeutung im Projekt
RTC	Real-Time Clock, hier das DS3231-Modul.
Snapshot	Ein einmal erfasster Zeitwert, aus dem mehrere zusammengehörige Felder gelesen werden.
Strobe / STR	Signal zur gemeinsamen Übernahme der Schieberegisterdaten in die sichtbaren Ausgänge.
TimeLib	Arduino-Zeitbibliothek, welche die aktive Systemzeit bereitstellt.

14.2 Verwendete Projektquellen

- DCF_Clock_V4a.ino, Programmstand und Kommentierung vom 03.08.2026.
- Schematic_DCF-Nixie-Clock_2026-06-29.png, Schaltbild der DCF-Nixie-Uhr.
- CD4049-signals.png, Messung von CLOCK, DATA und STROBE am unteren CD4094.
- Nixie Uhr Schaltungsanalyse.pdf, Analyse der Anzeigekette.
- NIXITRON_Softwarepruefung_DCF_Clock_V4.pdf, frühere Softwareprüfung.
- Ergebnisse der schrittweisen Hardwaretests im Projektverlauf.

Versionsbezug: Abweichende Verdrahtung sowie andere Bibliotheks- oder Arduino-Core-Versionen können das Verhalten verändern und müssen gesondert geprüft werden.