SLOTCAR STARTAMPEL

„Bären(n)keller“

▷ Software-Dokumentation

andreas wahl

Informationstechnik – Kommunikationstechnik – Elektronik
innovativ + kompetent + effizient

SLOTCAR STARTAMPEL

„Bären(n)keller“

Software-Dokumentation

Version 1.0 - 31.01.2024

Software-Version: 1.0
https://github.com/Andreas-Wahl/Startlight-SW

Hardware-Version: 1.3
https://github.com/Andreas-Wahl/Startlight-HW

Gehäuse-Version 1.0
https://github.com/Andreas-Wahl/Startlight-3D

andreas wahl

Informationstechnik – Kommunikationstechnik – Elektronik
innovativ + kompetent + effizient

https:/www.andreas-wahl.de/

Inhaltsverzeichnis

Einführung und Ziele

Idee und Namensgebung

In einer bereits über zwei Generationen bestehenden privaten Kellerbar, die ursprünglich den Namen „Bärenkeller“ erhalten hat, wurde eine CARRERA® digital 132 Rennbahn aufgebaut und die ersten Rennen ausgetragen.

Schnell fand dies bei Freunden und Bekannten großen Anklang. So wurde die Rennbahn sukzessive weiter ausgebaut und über mögliche Erweiterungen diskutiert.

So entstand auch die Idee einer eigenen Startampel, da die originale Startampel CARRERA® Startlight (30354) nicht über drei Fahrspuren (zwei Fahrspuren + Boxengasse) überbrücken kann. Weiter sollte die eigene STARTAMPEL nicht nur die Start-Sequenz über rote Anzeigen darstellen können, sondern auch den freigegebenen Rennbetrieb mit grünen LEDs, den Safety-Car-/Pace-Car-Betrieb und einen Frühstart anzeigen.

Kurzerhand wurde die Idee aufgegriffen, das Datensignal mit Oszilloskop und Datenlogger analysiert und die Entwicklung von Hardware, Software und Gehäuse-Design anzustoßen.

In Bezug auf den „Bärenkeller“, welcher durch die Erweiterung der CARRERA® digital 132 Rennbahn zum „Bärennkeller“ umbenannt wurde, erhielt auch die eigen entwickelte Startampel den Namen „STARTAMPEL „Bären(n)keller““.

Die STARTAMPEL „Bären(n)keller“ fand so großen Anklang, dass sich der Entwickler entschlossen hat, eine kleine Menge als „Prototypen“ zu produzieren und die Entwicklung für nicht kommerzielle Nutzung zur Verfügung zu stellen.

Die Elektronik und das Gehäusedesign sind unter der Creative Commons Version 4.0 Lizenz „Namensnennung, nicht kommerziell, Weitergabe unter gleichen Bedingungen“, (CC BY-NC-SA 4.0) und die Software unter der „GNU Affero General Public License, Version 3.0“ veröffentlicht.

Über diese Dokumentation

Ergänzend zu den Inline-Dokumentationen im Software-Quellcode soll diese an arc42[1] angelehnte Software-Dokumentation die Problemstellung beschreiben und eine mögliche Lösung der digitalen Signalverarbeitung von Manchester codierten Datensignalen anhand einer Carrera® digital 124/132 darstellen, sowie eine Möglichkeit der Realisierung einer finite state machine jeweils in Assembler für AVR®[2] Mikrocontroller aufzeigen.

Das Projekt „STARTAMPEL „Bären(n)keller“ soll insbesondere dazu anregen, sich spielerisch mit der Elektronik und der Programmierung von Mikrocontrollern, insbesondere in Assembler, auseinanderzusetzen.

Hierzu soll diese Dokumentation eine Hilfestellung bieten und weitere Anreize und Ideen bei interessanten Projekten bei Spiel und Spaß geben.

[1] arc42: https://arc42.org/
[2] AVR® Mikrocontroller: https://www.microchip.com/design-centers/8-bit/microchip-avr-mcus

Aufgabenstellung

Für die STARTAMPEL „Bären(n)keller“ wurden nachfolgend aufgeführte und beschriebe Funktionen festgelegt.

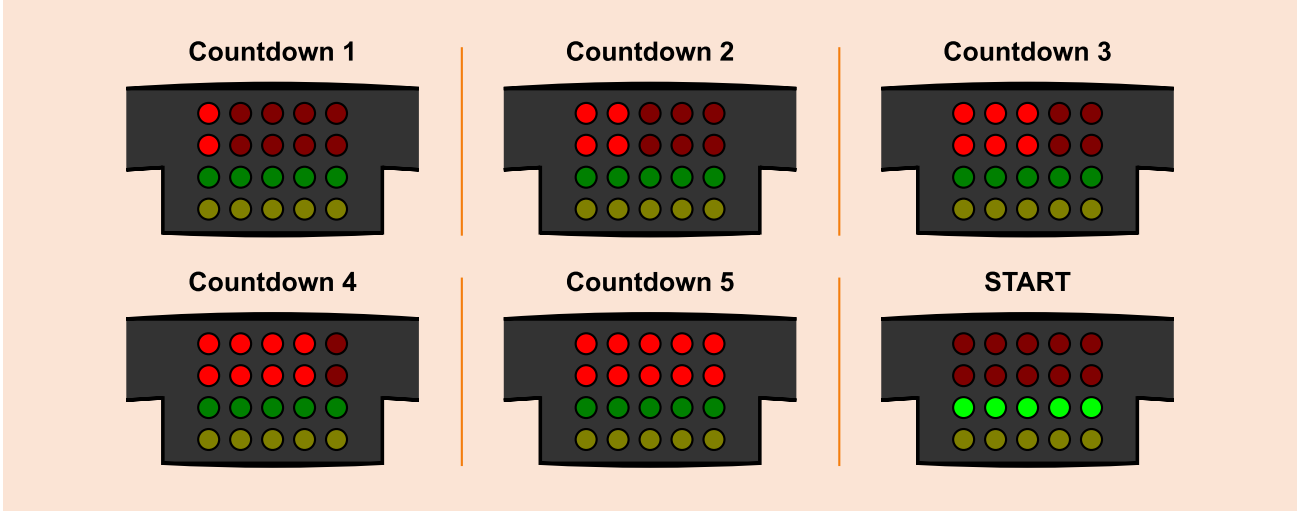
ID	Anforderung	Erklärung
M-1	Start-Sequenz	Der Countdown der Startsequenz soll über 5 rote LEDs, welche nacheinander angeschaltet werden, ausgegeben werden.
M-2	Früh-Start	Ein Frühstart soll durch dauerhaftes Leuchten der gelben LEDs signalisiert werden. Gleichzeitig ist der Verursacher über Blinken der entsprechenden Spalte der roten LEDs auszugeben.
M-3	PACE CAR	Bei aktivem PACE CAR sind die Rennteilnehmer über ein Blinken der gelben LEDs zu informieren.
M-4	CHAOS	Wird der Rennbetrieb unterbrochen (CHAOS), ist dies durch ein Blinken aller roten LEDs darzustellen.
M-5	Rennende / Sieger	Wird das Rennende über das Datenprotokoll empfangen, ist dies durch Blinken der grünen LEDs darzustellen. Über ein dauerhaftes Leuchten der entsprechenden roten LED-Spalte ist das Fahrzeug zu signalisieren, welches das Rennen beendet hat.
M-6	Rennbetrieb	Bei freigegebenem Rennbetrieb leuchten alle grünen LEDs dauerhaft.

Start-Sequenz:

Die grundlegende Funktion der STARTAMPEL „Bären(n)keller“ stellt die Startsequenz (Countdown) beim Rennstart dar.

Entsprechend des Countdowns sind die roten LEDs schrittweise von links nach rechts hinzuzuschalten.

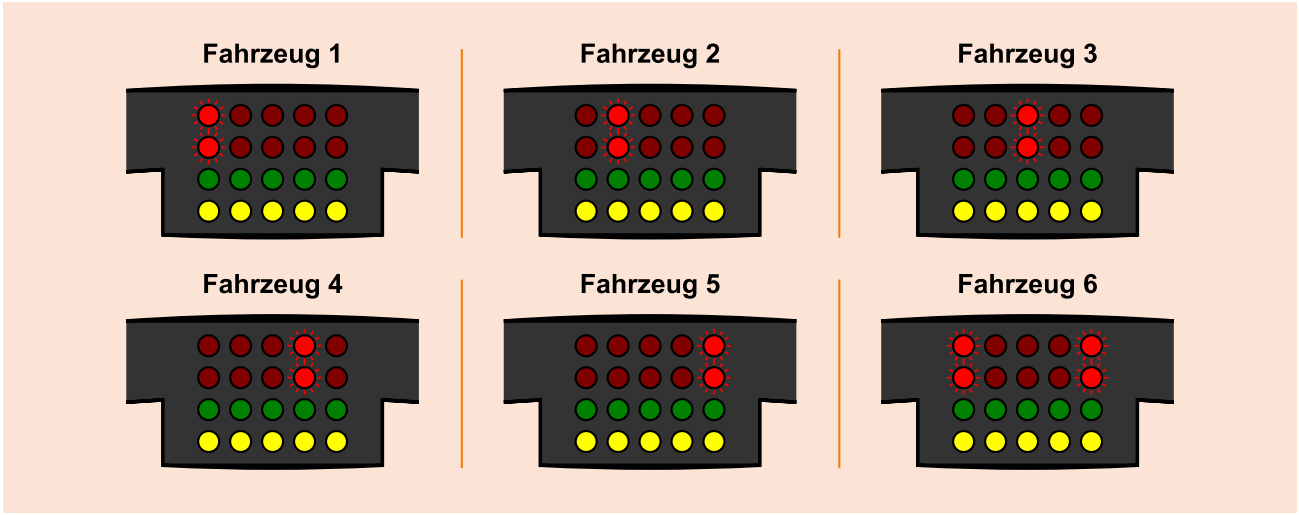
Bei Erreichen der Rennfreigabe sind die roten LEDs aus- und die grünen LEDs einzuschalten.



Früh-Start:

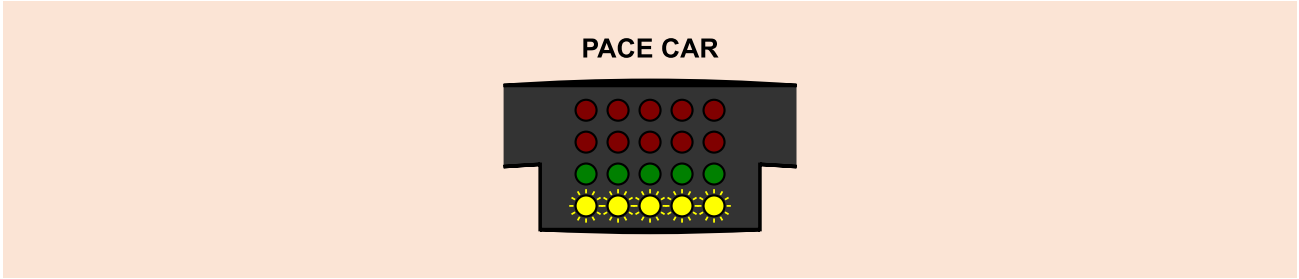
Erfolgt ein Frühstart während der Startsequenz, ist dieser durch die STARTAMPEL „Bären(n)keller“ mit Einschalten der gelben LEDs zu signalisieren.

Die Nummer des Fahrzeugs bzw. des Reglers, welcher den Früh-Start verursacht hat, ist über die roten LEDs blinkend ausgegeben.



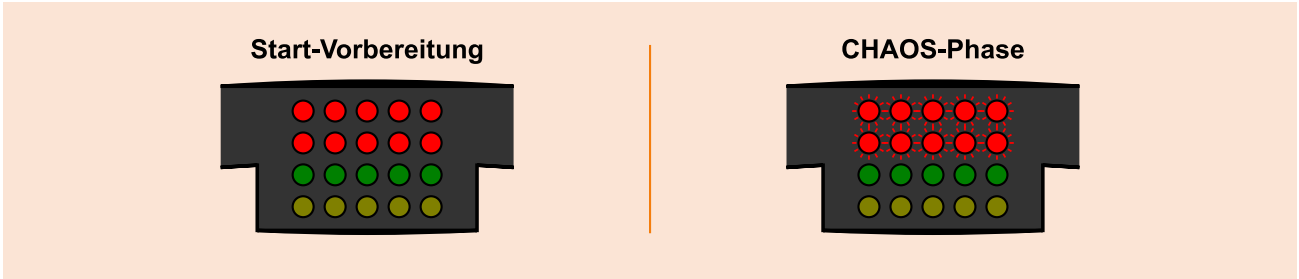
PACE CAR / SAFETY CAR

Während der PACE CAR- / SAFETY CAR-Phase ist diese durch die STARTAMPEL „Bären(n)keller“ durch Blinken der gelben LEDs zu signalisieren.



Renn-Unterbrechung (CHAOS) / Startvorbereitung

Eine Renn-Unterbrechung (CHAOS) / Streckensperrung wird durch die STARTAMPEL „Bären(n)keller“ während des Rennbetriebs – bzw. nach dem ersten Durchlauf der Startsequenz – durch Blinken aller roten LEDs zu signalisiert.

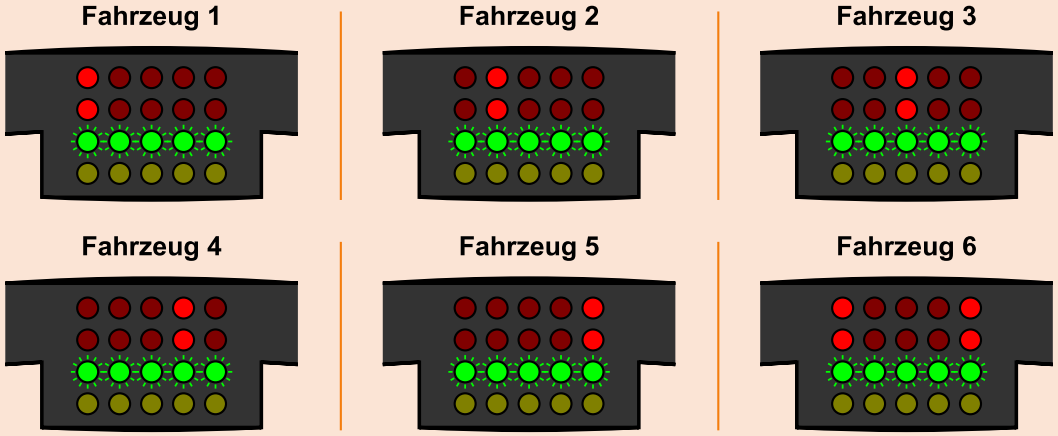


Wird kein aktives Rennen erkannt, bspw. nach Einschalten der CARRERA® Control-Unit[3] oder nach Rennende-Erkennung[4] wird die Unterbrechung als Startvorbereitung gewertet und mit dauerhaftem Leuchten aller 5 LEDs darzustellen.

[3] CARRERA® Control-Unit: https://carrera-toys.com/product/20030352-control-unit
[4] Nur mit CARRERA® Lap-Counter (https://carrera-toys.com/product/20030355-lap-counter) o. ä.

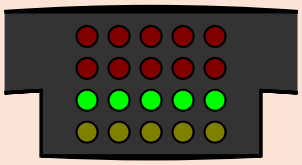
Renn-Ende / Sieger-Anzeige

Sofern das Rennende und der Sieger über die Datenleitung signalisiert und damit ausgewertet werden können, ist dies durch Blinken aller grünen LEDs darzustellen. Das Fahrzeug bzw. der Regler, welcher das Rennen beendet, bzw. gewonnen hat, ist über die roten LEDs auszugeben.[5]



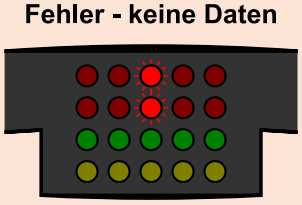
Rennbetrieb / Rennstrecke freigegeben

Der normale Rennbetrieb bzw. die freigegebene Rennstrecke wird durch die STARTAMPEL „Bären(n)Keller“ durch dauerhaftes Leuchten der grünen LEDs signalisiert.



Fehler – kein Datensignal

Erhält die STARTAMPEL „Bären(n)Keller“ kein gültiges Datensignal von der CARRERA® Control-Unit[6], ist dies mit dem Blinken der mittleren roten LED-Spalte zu signalisieren.



[5] Nur mit CARRERA® Lap-Counter (https://carrera-toys.com/product/20030355-lap-counter) o. ä.
[6] CARRERA® Control-Unit: https://carrera-toys.com/product/20030352-control-unit

Qualitätsziele

Folgende wesentlichen Qualitätsanforderungen wurden für die STARTAMPEL „Bären(n)keller“ bestimmt:

ID	Anforderung	Erklärung
Q-1	Korrektheit	Die jeweils signalisierten Zustände müssen korrekt sein.
Q-2	Zuverlässigkeit	Die Zustände müssen zuverlässig signalisiert werden.
Q-3	Leistungsfähigkeit	Schnelle Auswertung und kurze Reaktionszeit.
Q-4	Darstellung	Gute und verwechslungsfreie Darstellung der einzelnen Zustände.

Stakeholder

Folgende Personen und Gruppen haben Interesse an der Funktion und Qualität der STARTAMPEL „Bären(n)keller“:

Rolle	Ziel	Erwartungshaltung
Rennleitung	Zuverlässiger Betrieb.	Möchte eine zuverlässige und störungsfreie Funktion.
Fahrer	Schnelle, zuverlässige und fehlerfreie Signalisierung.	Möchte die Zustände des Rennbetriebes schnell, korrekt und zuverlässig erkennen können.
Zuschauer	Gute Visualisierung des Rennbetriebs.	Möchte die Zustände des Rennbetriebes gut und ohne Verwechslungsgefahr erkennen können.

Folgende Personen und Gruppen haben Interesse am Quellcode der Software, sowie der Dokumentation der STARTAMPEL „Bären(n)keller“:

Rolle	Ziel	Erwartungshaltung
Entwickler	Effizientes Umsetzen der definierten Anforderung in der Software.	Gute und Vollständige Beschreibung der zu Erwartenden Funktionen und Qualitätsziele, sowie umfangreiche Dokumentation der angebundenen Systeme.
Lektor	Prüfen der vorliegenden Dokumentationen und Quellcode.	Gute, verständliche und fehlerfreie Dokumentation der Anforderungen, Lösungsansätze und Quellcode.
Tester	Prüfen der Anwendung auf mögliche Fehler.	Software, die den definierten Anforderungen entspricht. Gute Bedienungsanleitung und Beschreibung der Funktionen.
Nutzer	Möchte den Quellcode gemäß der Lizenz weiterverwenden.	Fehlerfreien Quellcode, gute, verständliche und fehlerfreie Dokumentation.

Randbedingungen

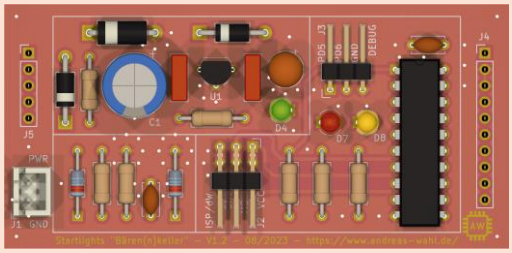
Für die Entwicklung der Software für die STARTAMPEL „Bären(n)keller“ liegen folgende Randbedingungen vor.

ID	Anforderung
RB-1	Elektronik: ▶ Microchip AVR®-Mikrocontroller ATtiny2313 ▶ 8 MHz Taktfrequenz, 16 bit und 8 bit Timer, externe Interrupts ▶ Vorgegebene Beschaltung
RB-2	Carrera® digital 124/132 Rennbahn[7] ▶ Datenübertragung in Anlehnung an den Manchester-Code ▶ CARRERA® Control-Unit[8] (20030352) ▶ CARRERA® Lap-Counter[9] (20030355)
RB-3	Programmierung in AVR® Assembler mit Microchip Studio für AVR®- und SAM-Devices[10]
RB-4	Universelle Decoder-Routine für alle CARRERA® Control-Unit Datenpakete
RB-5	Steuerung über einen Zustandsautomat (finite state machine)

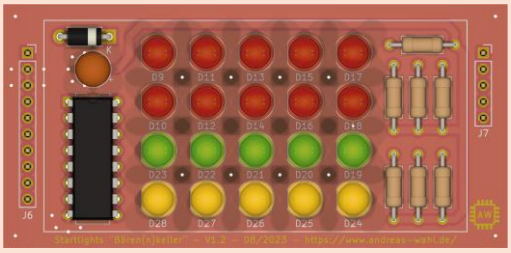
Hardware und Mikrocontroller

Für die STARTAMPEL „Bären(n)keller“ wurde eine eigene Hardware entwickelt.

Mikrocontroller



Display



<u>Hinweis:</u> Für die Elektronik (Hardware) wurde eine eigene Herdware-Dokumentation erstellt, auf die hier für detaillierte Informationen zur Elektronik (Hardware) verwiesen wird.

Technische Daten:

Zur Auswertung und Ansteuerung ist ein Microchip AVR®-Mikrocontroller ATtiny2313[11] gegeben.

Dieser wird mit einem internen Takt von 8 MHz betrieben. Das Fuse-Bit CKDIV8 wird nicht gesetzt, so dass ein 8 MHz Systemtakt zur Verfügung steht.

Der ATtiny2313 bietet einen 16 bit- und einen 8 bit-Timer. Zwei externe Interrupts, insgesamt 18 programmierbare Ein/Ausgänge,128 byte internen SRAM, 2 kbyte interner Flash-Speicher, ISP, debugWire und vieles mehr an.

---

[7] CARRERA® digital 132 Rennbahn: https://carrera-toys.com/digital-132
[8] CARRERA® Control-Unit: https://carrera-toys.com/product/20030352-control-unit
[9] CARRERA® Lap-Counter: https://carrera-toys.com/product/20030355-lap-counter
[10] Microchip Studio for AVR®: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio
[11] Microchip ATtiny2313: https://www.microchip.com/en-us/product/ATtiny2313

Beschaltung:

Port	Alias	Richtung	Beschreibung
PIND2	ioCMCsignal	Eingang (Ext.Int.0)	Datensignal Carrera® digital 124/132
PINB0	ioLEDyellow	Ausgang	gelbe LED-Reihe
PINB1	ioLEDgreen	Ausgang	grüne LED-Reihe
PINB2	ioLEDred5	Ausgang	rote LEDs Spalte 5
PINB3	ioLEDred4	Ausgang	rote LEDs Spalte 4
PINB4	ioLEDred3	Ausgang	rote LEDs Spalte 3
PINB5	ioLEDred2	Ausgang	rote LEDs Spalte 2
PINB6	ioLEDred1	Ausgang	rote LEDs Spalte 1

CARRERA® digital 124/132

Die Carrera® Control-Unit[12] der CARRERA® digital 124/132 Rennbahnen kommuniziert mit den Fahrzeugen mittels digitalem Datensignal in Anlehnung an den Manchester-Code[13].

Dabei wird die Fahrspannung für die Datenübertragung entsprechend auf 0 V (GND) gezogen.

Nachdem zunächst mit Oszilloskop und Datenlogger die Datenpakete der CARRERA® Control-Unit analysiert wurden, hatten zusätzliche Recherchen im Internet unter anderem auf die Internetseite von Stephan Heß[14] geführt, welcher bereits eine umfangreiche Analyse der CARRERA® digital 124/132 Datenübertragung durchgeführt und auf seiner Internetseite veröffentlicht hat.

Verschiedene Bezeichnungen zur CARRERA® Datenübertragung wurden von Stephan Heß übernommen. Auch konnte die eigene Analyse des Datenprotokolls mit seinen Analysen abgeglichen und ergänzt werden.

Programmiersprache und Entwicklungsumgebung

Die Software ist in der Programmiersprache „Assembler“ zu entwickeln.

Aufgrund der Tatsache, dass aktuell viele veröffentlichte Mikrocontroller-Projekte nahezu ausschließlich in Hochsprachen – insbesondere im „Hobby“- und „Schulbereich“ meist mit Arduino Sketch oder ähnlichen – entwickelt werden, soll hier gezeigt werden, wie verschiedene Aufgaben auch in der maschinennahen Programmiersprache „Assembler“ gelöst werden können.

Entwicklungsumgebung: Microchip/Atmel Studio für AVR® and SAM Devices – Version 7.0[15]

Der Manchester-Decoder soll alle Carrera® digital 124/132 Datenpakete auswerten und zur weiteren Verarbeitung zur Verfügung stellen können.

Die Steuerung der einzelnen Zustände der STARTAMPEL „Bären(n)keller“ soll über einen Zustandsautomat (finite state machine, FSM) erfolgen.

Der Grundaufbau der Manchester-Dekodierung und des Zustandsautomaten ist jeweils so aufzubauen, dass dieser auch auf andere ähnliche Projekte übertragen werden kann.

---

[12] CARRERA® Control-Unit: https://carrera-toys.com/product/20030352-control-unit
[13] Manchester Code: https://de.wikipedia.org/wiki/Manchester-Code
[14] Stephan Heß: http://www.slotbaer.de/
[15] Microchip Studio for AVR®: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio

Kontextabgrenzung

Fachlicher Kontext

CARRERA® digital 124/132 – Datenpakete

Die Datenübertragung erfolgt mit einer Taktfrequenz von 10 kHz (10.000 Baud) in mehreren Datenpaketen in zyklischer Reihenfolge. Die Datenpakete haben je nach Typ unterschiedliche Längen (Bits). Der Abstand der einzelnen Datenpakete beträgt jeweils 7,5 ms.

Nr.	Datenpaket	Länge (Bit)
1	Control-Unit (CU)	12
2	PACE CAR / GHOST CAR (PGC)	9
3	Aktive Fahrzeuge/Regler Quittierung	7 8
4	Fahrzeug/Regler 1	9
5	Fahrzeug/Regler 5	9
6	Fahrzeug/Regler 2	9
7	Fahrzeug/Regler 6	9
8	Fahrzeug/Regler 3	9
9	Aktive Fahrzeuge/Regler	7
10	Fahrzeug/Regler 4	9

Das Control-Unit-Datenpaket hat eine Länge von 12 Bit (3 Nibble) und ist das längste Datenpaket. Über die Länge des Control-Unit-Datenpakets lässt sich der Start der zyklisch wiederholenden Datenpakete erkennen und so die einzelnen nachfolgenden Datenpakete ihrer Funktion zuordnen.

Für die Realisierung der STARTAMPEL „Bären(n)keller“ sind ausschließlich die beiden Datenpakete „Control-Unit“ und „PACE CAR / GHOST CAR“ relevant.

Control-Unit (CU) – Datenpaket:

Das Control-Unit-Datenpaket im Format little endian hat eine Länge von 12 Bit und enthält nachfolgende Informationen:

Wert				**Instruktion**					**Adresse**		
4 Bit				5 Bit					3 Bit		
0	1	2	3	0	1	2	3	4	0	1	2
W	**W**	**W**	**W**	**I**	**I**	**I**	**I**	**I**	**A**	**A**	**A**

Das LSB wird beim Control-Unit-Datenpaket jeweils zuerst übertragen (LSb-first)!
→ Alle Bits stehen in umgekehrter Reihenfolge im Register.

Start-Sequenz (rote LEDs):

```
<Wert=Countdown/LED><Instruktion=16><Adresse=7 (CU)>

WWWW00001111
```

Frühstart:

```
<Wert=1><Instruktion=11><Adresse=Fahrzeug/Regler>

100011010AAA
```

Rennende / Sieger:

```
<Wert=1><Instruktion=7><Adresse=Fahrzeug/Regler>

100011100AAA
```

PACE CAR / GHOST CAR (PGC)– Datenpaket:

Das PACE CAR-Datenpaket im Format big endian hat eine Länge von 9 Bit und enthält nachfolgende Informationen:

Adresse			**Zustand/Flag**					
3 Bit			6 Bit					
2	1	0	5	4	3	2	1	0
A	**A**	**A**	**F5**	**F4**	**F3**	**F2**	**F1**	**F0**
=7 (PACE CAR)			**Strecke gesperrt**	**Takt(?)**	**Strecke offen**	**PACE CAR Rückruf(?)**	**PACE CAR aktiv**	**Spritverbrauch aktiviert**

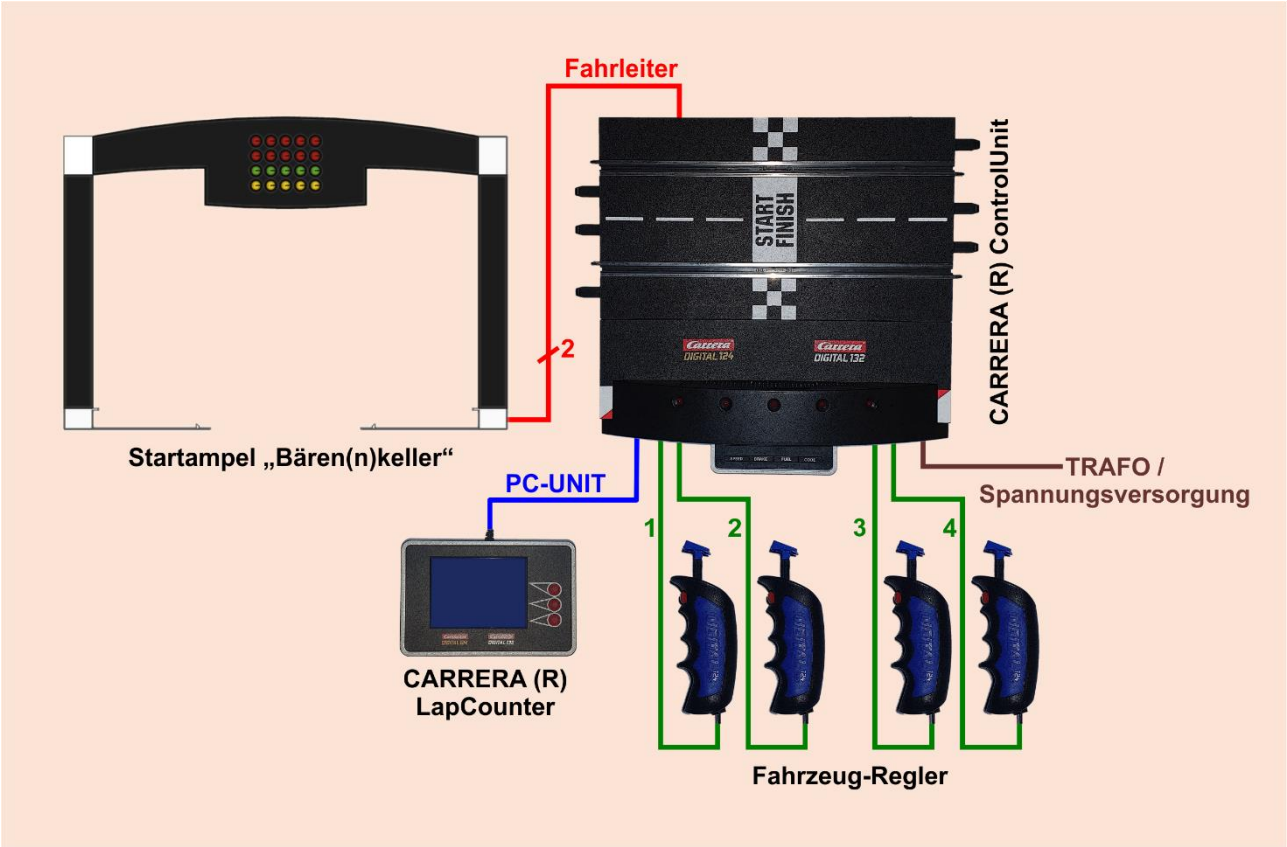
Beim PACE CAR-Datenpaket wird das MSB zuerst übertragen (MSb-first)!
→ Alle Bits stehen in korrekter Reihenfolge in den Registern.

PACE CAR:

```
<Adresse=7 (PACE CAR)><F5><F4><F3><F2><F1=PACE CAR aktiv><F0>

111xxxxPx

111xxxx1x   --> PACE CAR ist aktiv

111xxxx0x   --> PACE CAR ist inaktiv
```

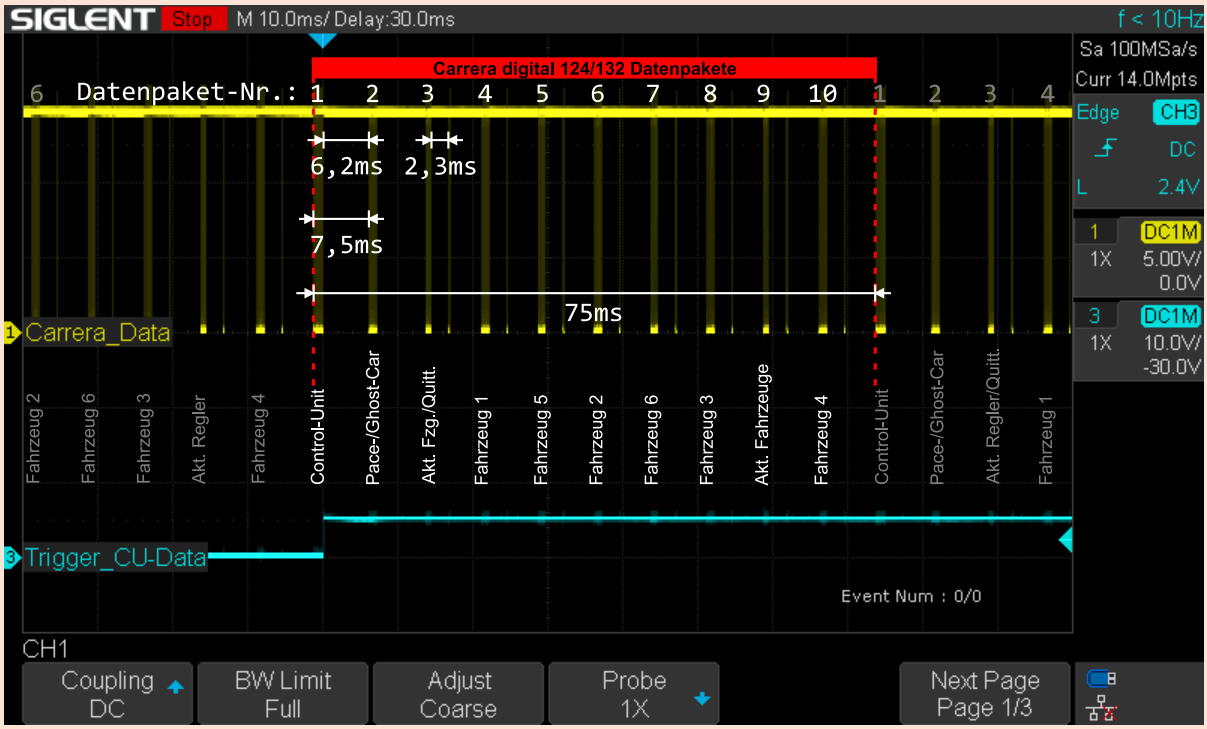
Technischer Kontext

Übersicht der Komponenten

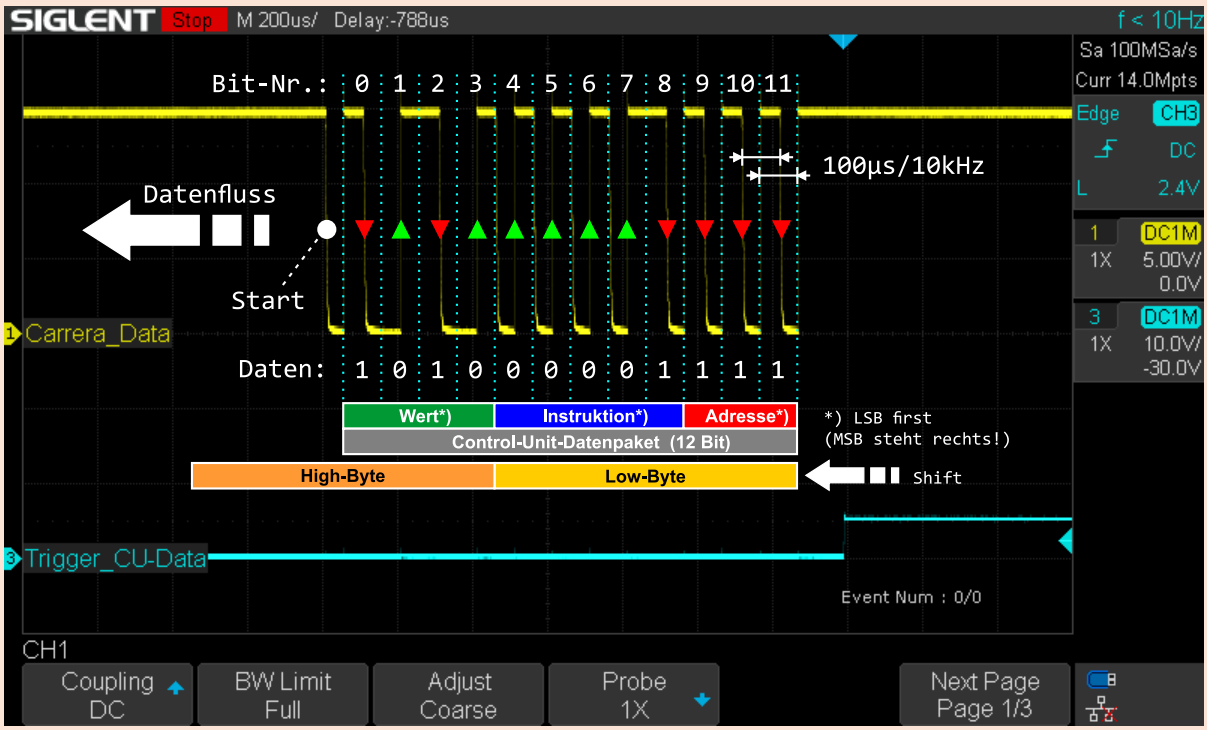


Bezeichnung	Beschreibung
Control-Unit	Zentraleinheit der CARRERA® digital 124/132 Rennbahn für die Bedienung und Steuerung des Rennbetriebs und Anschluss aller Komponenten.
Lap-Counter	Rundenzähler für die CARRERA® digital 124/132 Rennbahn. Über Ihn werden Zeit und Rundenrennen gestartet und das Rennen entsprechend beendet, wenn Zeit oder Anzahl der Runden erreicht wurden. Der Sieger, bzw. das Fahrzeug, welcher(s) das Rennen beendet, wird durch den Lap-Counter signalisiert. Alternativ können an der PC-UNIT Schnittstelle auch anderweitige Lösungen, wie bspw. PC-Software angebunden werden.
Regler	Mit den Reglern (auch Controller oder Drücker genannt) werden die Fahrzeuge (fern-)gesteuert.
Fahrzeug	Das Fahrzeug (Slotcar) im Maßstab 1:24 bzw. 1:32 fährt mittels Leitkiel in einem Schlitz seiner zugehörigen Fahrspur auf der Rennstrecke. Über Kontaktschleifer wird das Fahrzeug über die Fahrbahn mit Spannung versorgt. Bei der CARRERA® digital 124/132 Rennbahn erhält das Fahrzeug über den Fahrleiter auch die Steuerinformationen mittels Datenpaketen im Manchester-Code.
Startampel	Die STARTAMPEL „Bären(n)keller“ ergänzt die Rennstrecke um eine Visualisierung der Start-Sequenz, des Zustandes der Rennstrecke usw.
Datenpakete	Mittels Datenpaketen, welche im Manchester-Code über den Fahrleiter gesendet werden, kommuniziert die CARRERA® digital 124/132 Control-Unit mit Fahrzeugen und anderen Komponenten wie Weichen, Zeitmessungen, Boxengassen u. ä. Auch die für die STARTAMPEL „Bären(n)keller“ erforderlichen Informationen werden so übertragen.

CARRERA® digital 124/132 Datenpakete:



CARRERA® digital 124/132 Control-Unit-Datenpaket:



Zur Datenanalyse wurde jeweils das Oszilloskop SIGLENT SDS1204X-E verwendet. Mittels Bildbearbeitungsprogramm wurden die Oszilloskop-Ausdrucke um weitere Informationen ergänzt.

Lösungsstrategie

Die Software wird in folgende Funktionsteile gegliedert:

- **MAIN**
  das Hauptprogramm, enthält die Startsequenz, die Hauptschleife (main-loop) und enthält die ausgegliederten Funktionen per include-Direktive.
- **DECODER**
  enthält alle für die Dekodierung und Auswertung erforderliche Programmteile
- **FSM (finite state machine / Zustandsautomat[16])**
  übernimmt die Steuerung der STARTAMPEL „Bären(n)keller“.

Decoder

Manchester-Code[17]

Der Manchester-Code wird gebildet, indem das ursprüngliche Datensignal mit dem Taktsignal exklusiv verodert (XOR) wird. Die Information ist somit im Flankenwechsel in der Mitte einer Periode des Taktsignales enthalten.

Da der Fahrleiter mit +14,8 V bzw. +18 V zur Spannungsversorgung der Fahrzeuge dient, kann der Manchester-Code nicht gleichanteilsfrei übertragen werden. Deshalb wird bei der Datenübertragung der Fahrleiter entsprechend nach Ground (0 V) gezogen. Die Übertragung des Startbits beginnt mit einer fallenden Flanke der Fahrspannung nach Ground. Dies stellt das Startbit dar. Die Datenübertragung erfolgt mit 10.000 Baud, also mit einer Datenübertragungsfrequenz von 10 kHz. Jede folgende Flanke bei 1 τ (hier 100 µs) entspricht damit einem neuen Bit.

Ein Flankenwechsel von 5 V → 0 V am Porteingang des Mikrocontrollers entspricht einer logischen „1“, ein Flankenwechsel von 0 V → 5 V dementsprechend einer logischen „0“.

Für die Auswertung wird der externe Interrupt-Eingang, an dem der Manchester-Code anliegt, so konfiguriert, dass dieser bei jedem Flankenwechsel „ausgelöst“ wird.

Über einen Zähler innerhalb eines Timer-Overflow-Interrupts wird die Zeit zwischen den Flankenwechseln geprüft. Liegt dieser zwischen 0,75 τ und 1,25 τ (hier ~80 µs und ~120 µs) wird ein neues Bit empfangen. Der Pegel des Eingangssignals nach erfolgtem Flankenwechsel kann dann in ein Empfangsregister geschoben werden (LOW entspricht 1, HIGH entspricht 0).

Erfolgt nach 1,25 τ (hier ~125 µs) kein erneuter Flankenwechsel, ist das Ende des Datenpaketes erreicht (Frame-Ende). Nach Speicherung im SRAM kann die Auswertung bzw. Verarbeitung der empfangenen Datenpakete erfolgen.

Weitere Informationen zur Kodierung und Dekodierung des Manchester-Code mit Microchip AVR®-Mikrocontroller können aus den Atmel Application Notes – 9164 – Manchester Coding Basics[18] entnommen werden.

Speicherung (SRAM)

Die empfangenden Datenpakete (Speichergröße: 2 byte) werden in deren Reihenfolge entsprechend im SRAM abgelegt.

Für eine mögliche Plausibilitätsprüfung wird zusätzlich die Position/Nummer (Speichergröße: 1 byte) des Datenpaketes und die Länge in Bit (Speichergröße: 1 byte) des Datenpaketes gespeichert.

---

[16] Zustandsautomat: https://de.wikipedia.org/wiki/Endlicher_Automat
[17] Manchester Code: https://de.wikipedia.org/wiki/Manchester-Code
[18] Atmel Manchester Coding Basics: https://ww1.microchip.com/downloads/en/Appnotes/Atmel-9164-Manchester-Coding-Basics_Application-Note.pdf

Daten-Übersicht:

Symbol (.equ)	SRAM-Adresse	Daten
pOffsetDpPos	+00h	**Position** des Datenpakets
pOffsetDpCnt	+01h	**Länge** des Datenpakets in Bit
pOffsetDpLow	+02h	**Low-Byte** des Datenpakets
pOffsetDpHigh	+03h	**High-Byte** des Datenpakets

Datenpaket-Übersicht:

Symbol (.equ)	SRAM-Adresse	Datenpaket
pStartProgCU	+00h	**Control-Unit (CU)**
pStartPgc	+04h	**PACE CAR / GHOST CAR (PGC)**
pStartAckn	+08h	**Aktive Regler Quittierung**
pStartCtrl0	+0Ch	**Fahrzeug/Regler 1**
pStartCtrl4	+10h	**Fahrzeug/Regler 5**
pStartCtrl1	+14h	**Fahrzeug/Regler 2**
pStartCtrl5	+18h	**Fahrzeug/Regler 6**
pStartCtrl2	+1Ch	**Fahrzeug/Regler 3**
pStartActive	+20h	**Aktive Regler**
pStartCtrl3	+24h	**Fahrzeug/Regler 4**
(pStartDebug)	*(+28h)*	***(DEBUG)***

Auswertung:

Sobald ein Datenpaket erfolgreich empfangen und im SRAM gespeichert wurde, werden diese über eine Auswerte-Routine ausgewertet und die Ergebnisse in einem 8 Bit-Register für die Übergabe (rTransfer) an den Zustandsautomaten (finite state machine) aufbereitet und übergeben.

Zustand/Flag					**Wert**		
5 Bit					3 Bit		
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RESET (flProgReset)	**PACE CAR** (flProgPaceCar)	**Frühstart** (flProgEarlyStart)	**Start-Sequenz** (flProgStartLights)	**Rennende** (flProgFinished)	**Wert** (mskProgValues)		

Zustandsautomat (finite state machine)

Die STARTAMPEL „Bären(n)keller“ kann ausschließlich fest definierte Zustände einnehmen, deren Eintritts- bzw. Austrittsbedingungen fest definiert sind. Die Steuerung mittels Zustandsautomat[19] (finite state machine) ist daher obligatorisch.

Zustands-Tabelle:

Aktueller Zustand	Ampel Steuerung	Eingabe (Austrittsbedingung)	Nächster Zustand
0-Init	Grün: blinken andere: aus	Rennstrecke Frei 5 rote LED ein (Countdown 5)	6-Grün 5-Rot5
1-Rot1	Rot 1: ein andere: aus	2 rote LED ein (Countdown 2) 5 rote LED ein (Countdown 5) Frühstart	2-Rot2 5-Rot5 8-Frühstart
2-Rot2	Rot 1+2: ein andere: aus	3 rote LED ein (Countdown 3) 5 rote LED ein (Countdown 5) Frühstart	3-Rot3 5-Rot5 8-Frühstart
3-Rot3	Rot 1-3: ein andere: aus	4 rote LED ein (Countdown 4) 5 rote LED ein (Countdown 5) Frühstart	4-Rot4 5-Rot5 8-Frühstart
4-Rot4	Rot 1-4: ein andere: aus	5 rote LED ein (Countdown 5) Frühstart	5-Rot5 8-Frühstart
5-Rot5	Rot 1-5: ein andere: aus	1 rote LED ein (Countdown 1) Rennstrecke Frei Frühstart	1-Rot1 6-Grün 8-Frühstart
6-Grün	Grün: ein andere: aus	5 rote LED ein (Countdown 5) Chaos (5 rote LED + Rennen läuft) Frühstart PACE CAR aktiv Rennende	5-Rot5 10-Chaos 8-Frühstart 7-PaceCar 9-Rennende
7-PaceCar	Gelb: blinken andere: aus	PACE CAR inaktiv	6-Grün
8-Frühstart	Gelb: ein Rot 1-5 blinken gemäß Fahrzeug. Grün aus	5 rote LED ein (Countdown 5)	5-Rot5
9-RennEnde	Grün: blinken Rot 1-5 ein gemäß Fahrzeug. Gelb aus	5 rote LED ein (Countdown 5)	5-Rot5
10-Chaos	Rot 1-5 blinken andere: aus	Rennstrecke Frei 1 rote LED ein (Countdown 1)	6-Grün 1-Rot1

Bei RESET Austritt aus allen Zuständen zu 0-Init.

[19] Zustandsautomat: https://de.wikipedia.org/wiki/Endlicher_Automat

Indirekte Sprünge zum Zustand:

Die unterschiedlichen Zustände werden über indirekte Sprünge in Abhängigkeit der Statusnummer aufgerufen; nachfolgend beispielhaft in einem Quellcode-Beispiel dargestellt.

Sprungtabelle „entry“

```
    ; FSM jump table - entry state.
FSM_pStateEntry:
    rjmp    FSM_10State0_0Entry          ; State No. 0
    rjmp    FSM_20State1_0Entry          ; State No. 1
    rjmp    FSM_30State2_0Entry          ; State No. 2
```

Sprungtabelle „run“

```
    ; FSM jump table - running state.
FSM_pStateRun:
    rjmp    FSM_10State0_1Run            ; State No. 0
    rjmp    FSM_20State1_1Run            ; State No. 1
    rjmp    FSM_30State2_1Run            ; State No. 2
```

Aufruf des Zustandsautomaten- run:

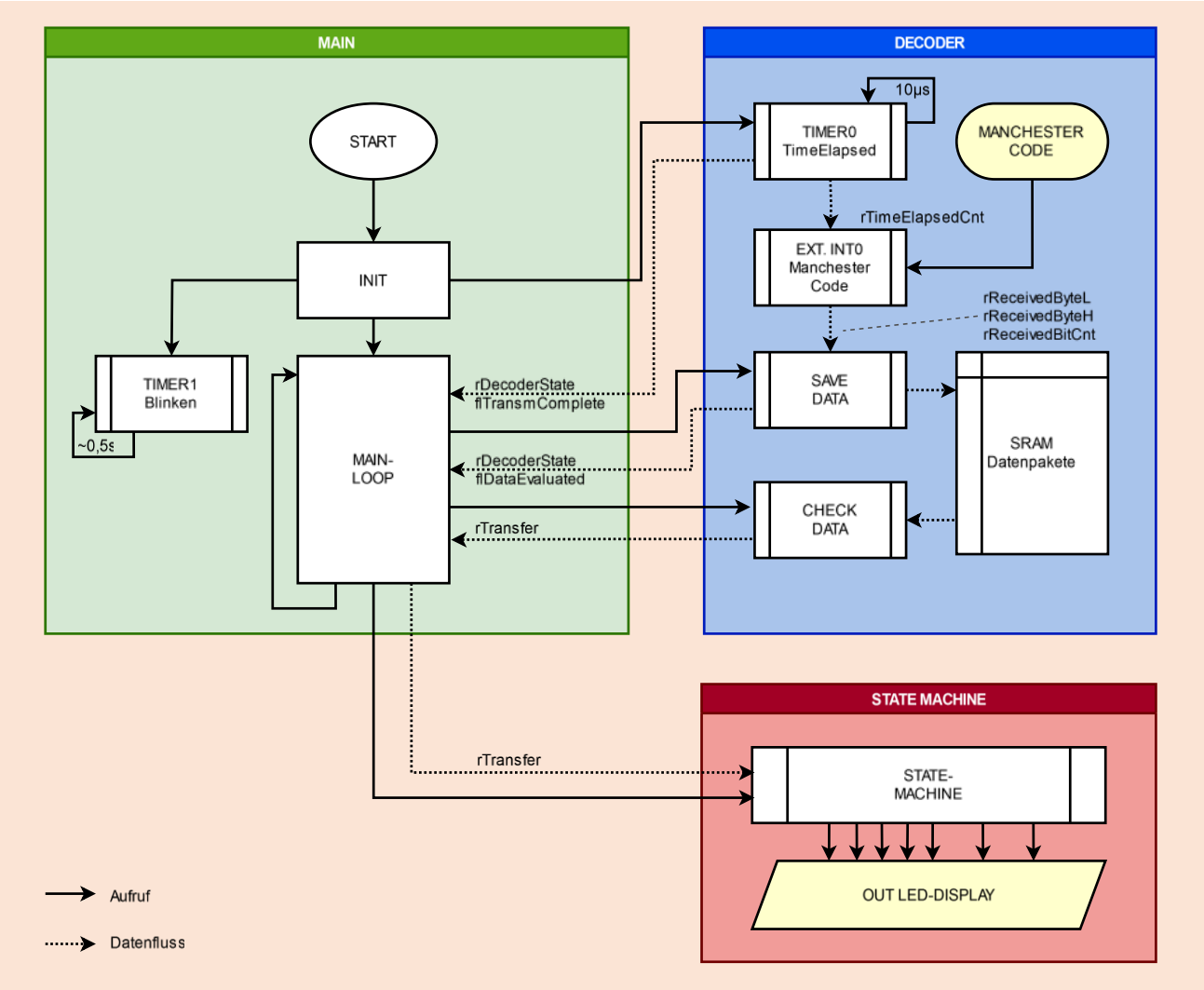
```
    ;  FINITE STATE MACHINE (FSM) - run
sub_FSM_00run:
    ; (Save SREG and registers to STACK, ...)
    ; (rFsmNextState=rFsmCurrentState, ...)
    ; Jump to current state.
    ldi     ZH, HIGH (FSM_pStateRun)     ; Load jump table address...
    ldi     ZL, LOW (FSM_pStateRun)      ; ...to pointer.
    add     ZL, rFsmCurrentState         ; Add state offset
    clr     rAccu
    adc     ZH, rAccu                    ; Consider carry flag
    ijmp                                 ; Jump to current state
```

Zustandswechsel - entry:

```
    ; FSM change state / next state - entry
FSM_98ChangeState:
    ; Jump to next state.
    ldi     ZH, HIGH (FSM_pStateEntry)   ; Load jump table address...
    ldi     ZL, LOW (FSM_pStateEntry)    ; ...to pointer.
    add     ZL, rFsmNextState            ; Add state offset
    clr     rAccu
    adc     ZH, rAccu                    ; Consider carry flag
    ijmp                                 ; Jump to new state entry
```

Baustein- und Verteilungssicht

Übersicht

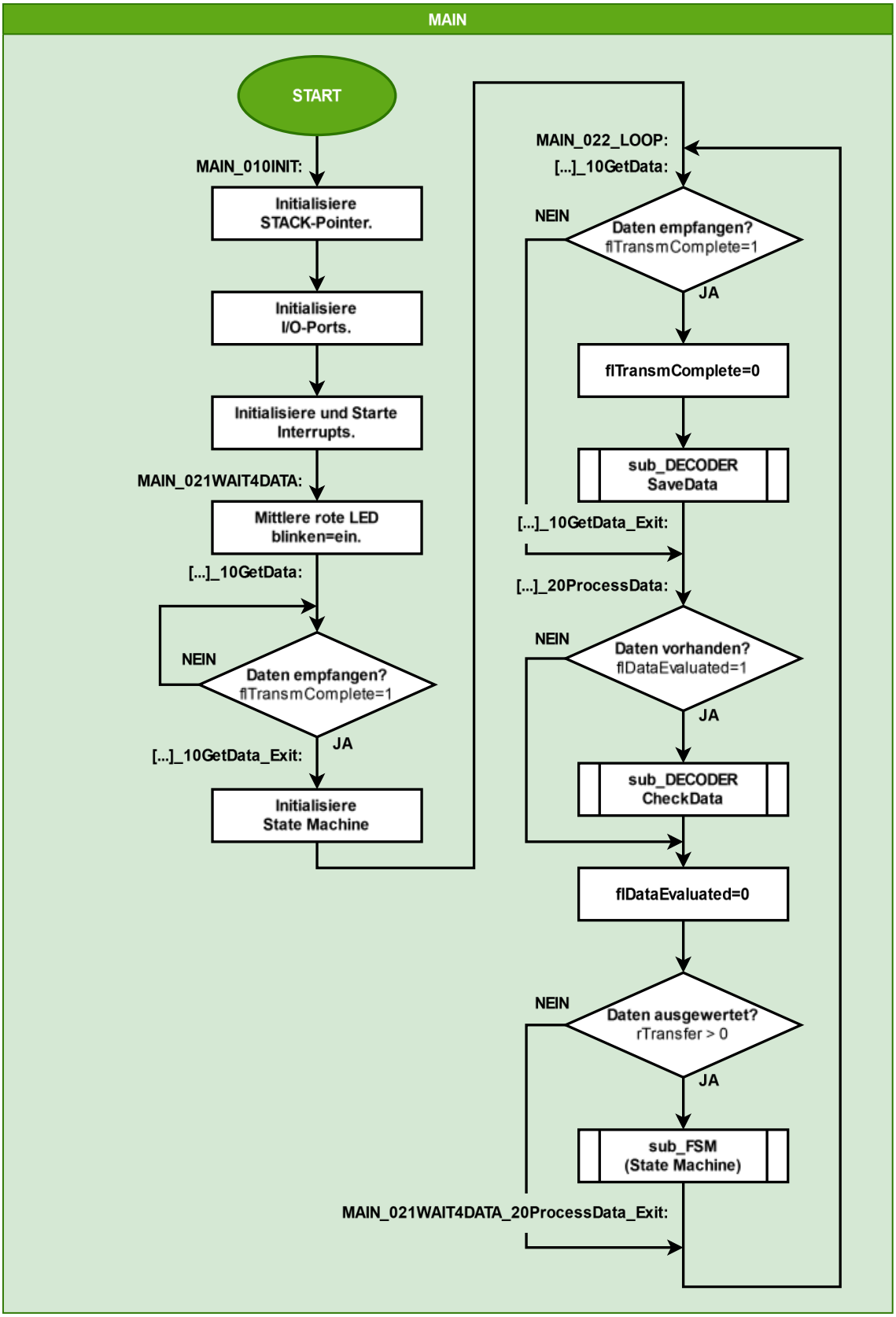


Bezeichnung	Beschreibung
Hauptprogramm (main)	Das Hauptprogramm initialisiert alle Komponenten und steuert mit seiner Hauptschleife (MAIN-LOOP) das Programm.
Auswertung (decoder)	Enthält alle für die Dekodierung und Auswertung des CARRERA® digital 124/132[20] Datensignals erforderliche Programmteile.
Steuerung (state machine)	Steuert den Zustand der STARTAMPEL „Bären(n)keller“ und gibt diesen über die I/O-Ports des Mikrocontrollers an den LED aus.

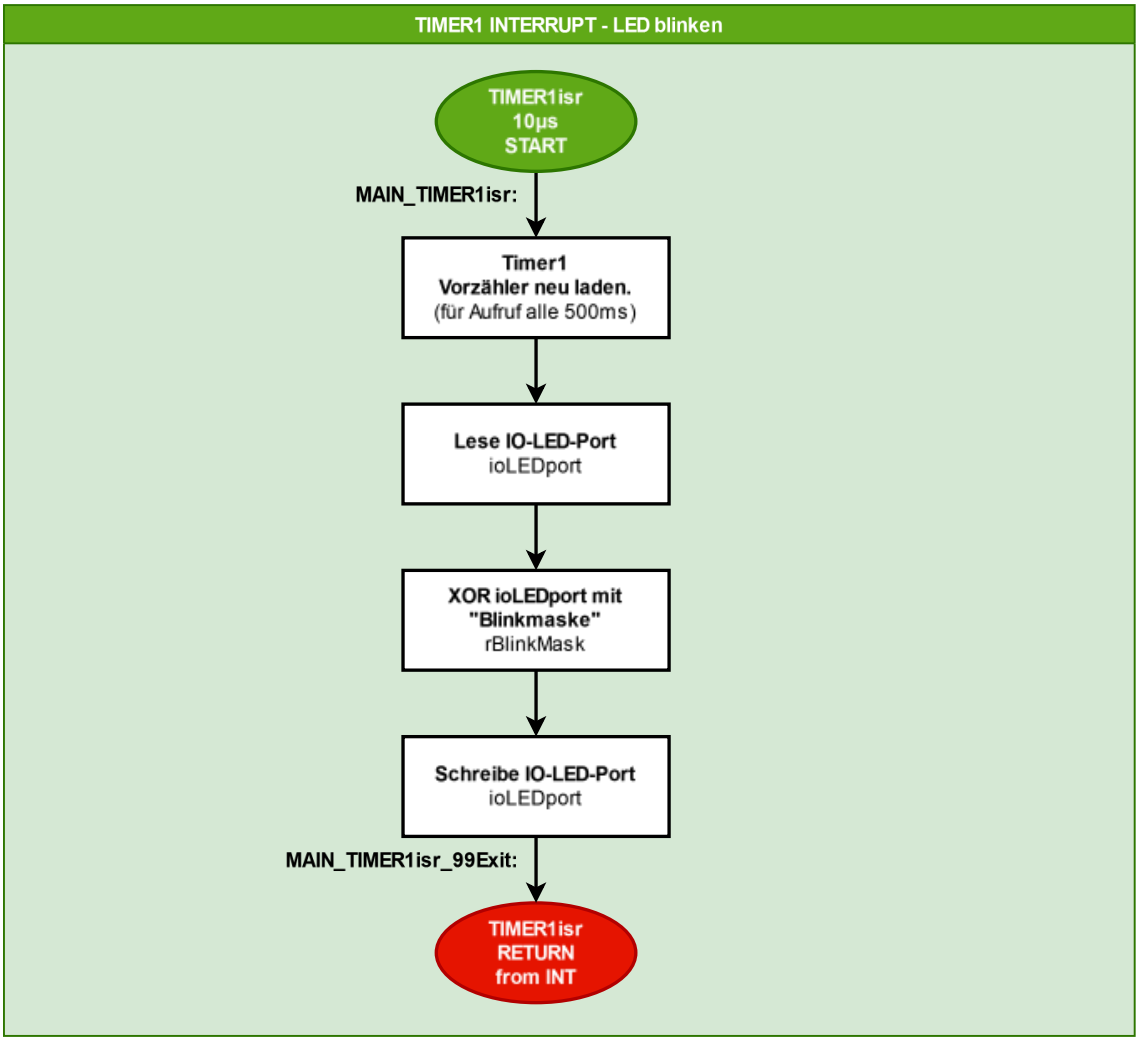
[20] CARRERA® digital 132 Rennbahn: https://carrera-toys.com/digital-132

Programm-Ablaufpläne

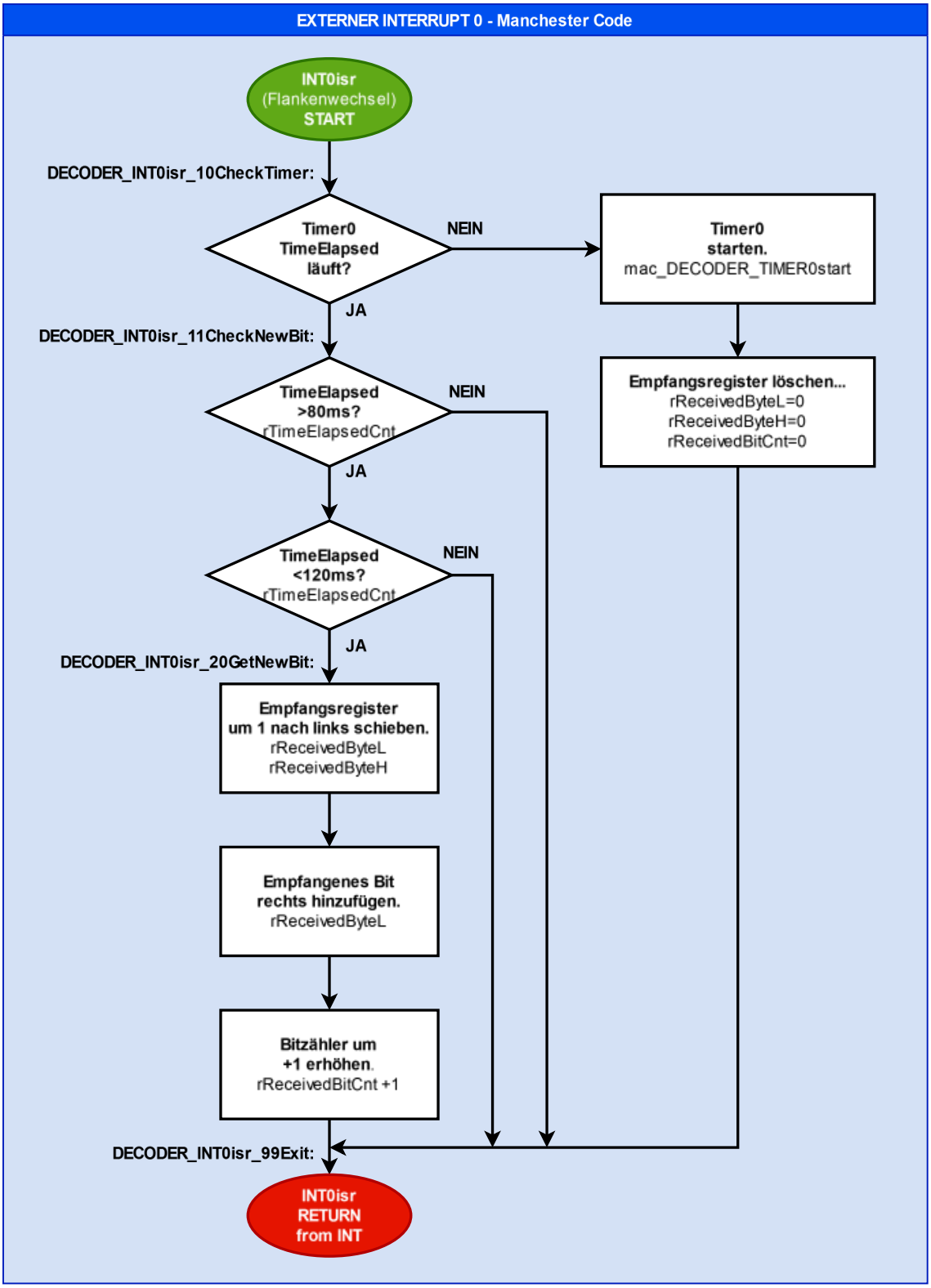
MAIN: Hauptprogramm



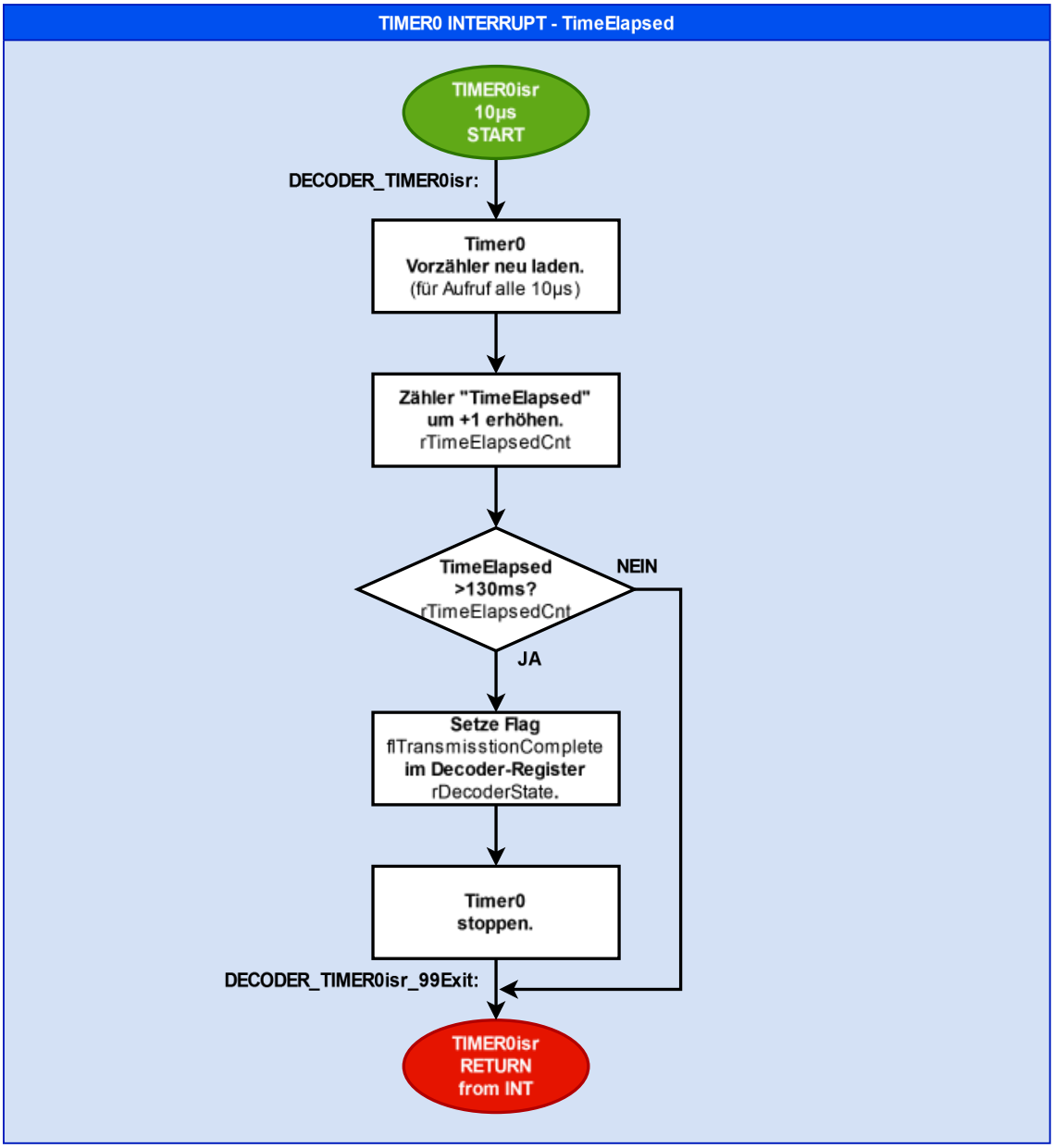
MAIN: Timer1 – LED-Blinken



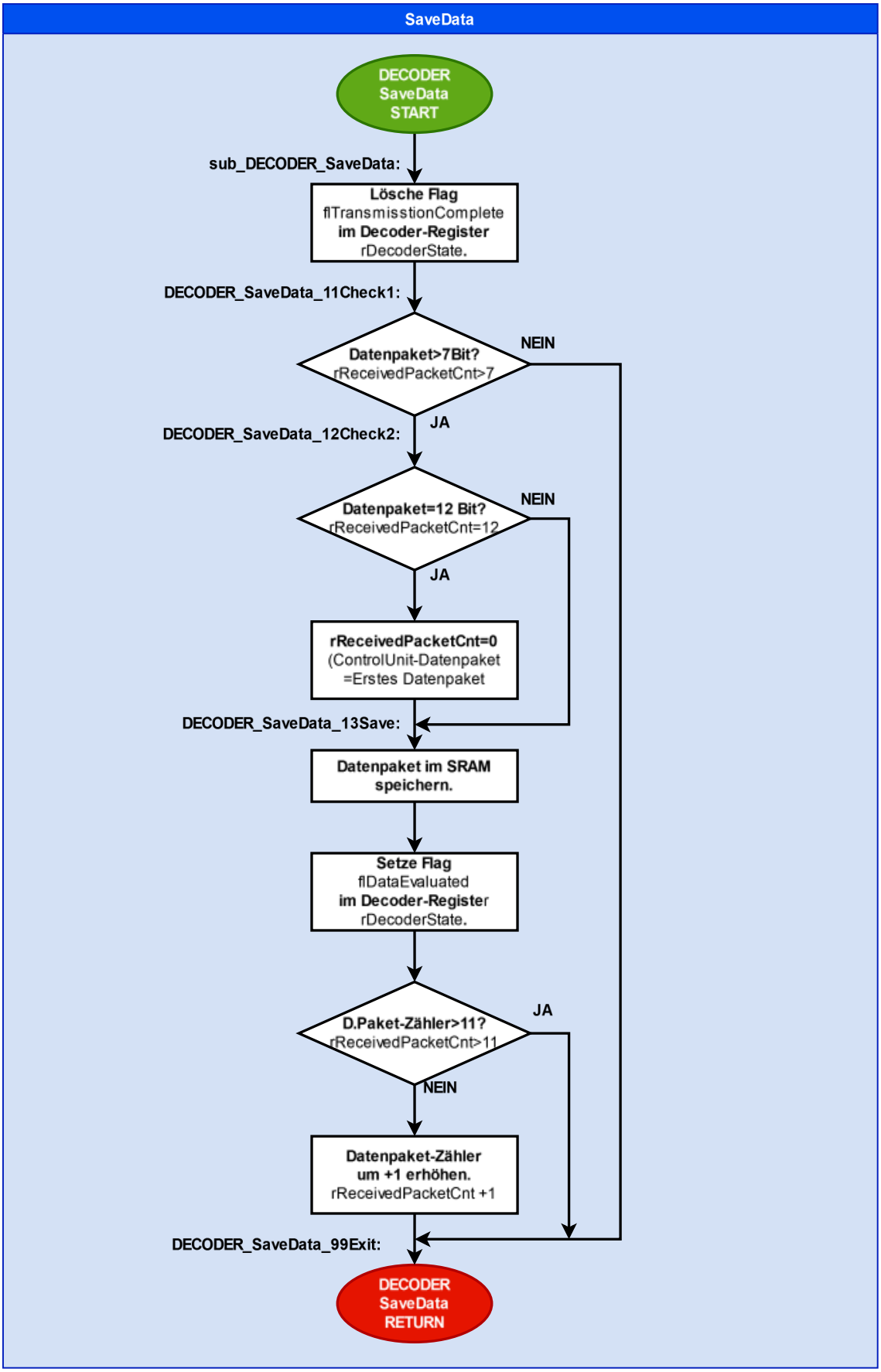
DECODER: ExtInt0 – Manchester Signal



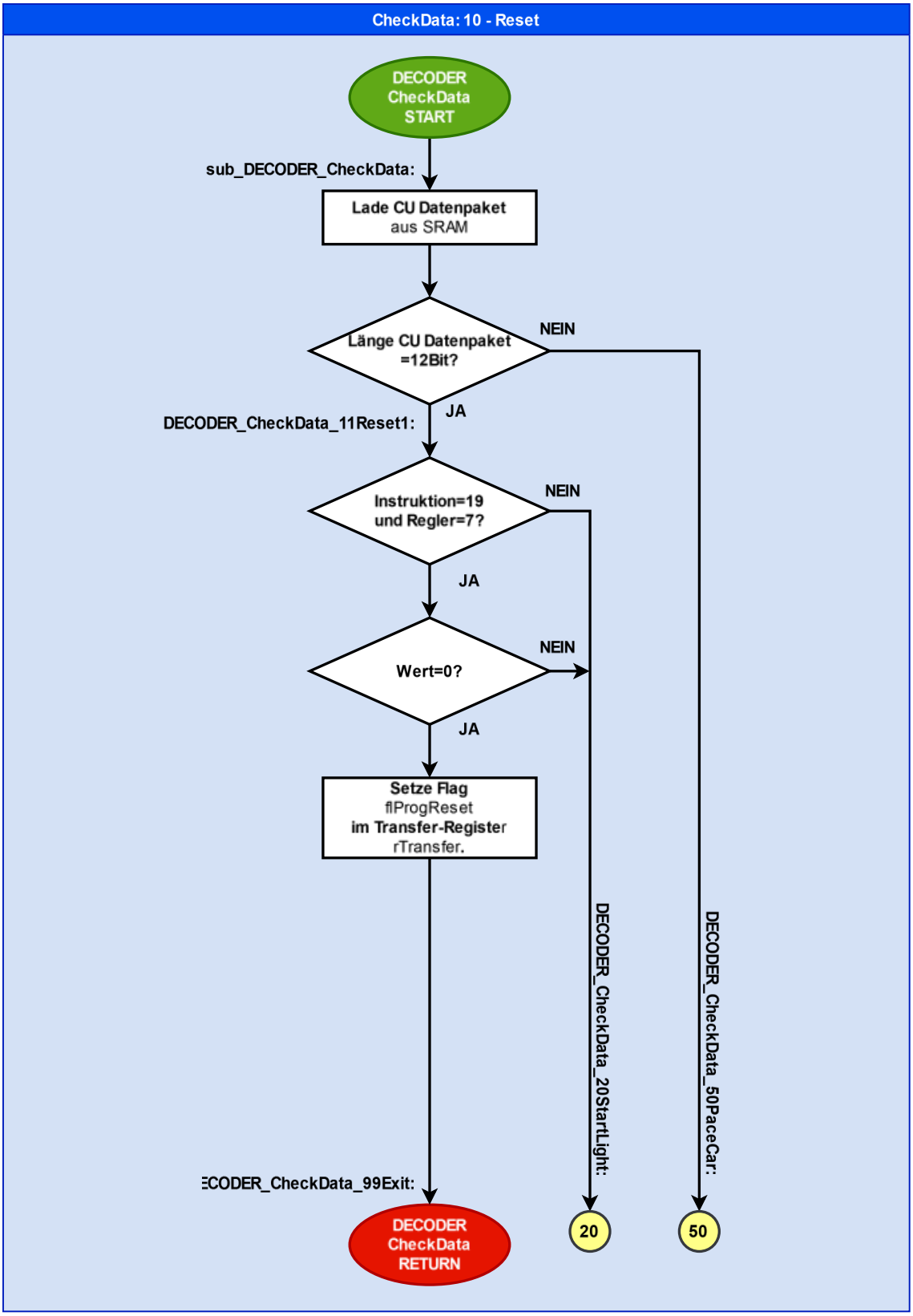
DECODER: Timer0 – TimeElapsed – Zähler für vergangene Zeit in µs



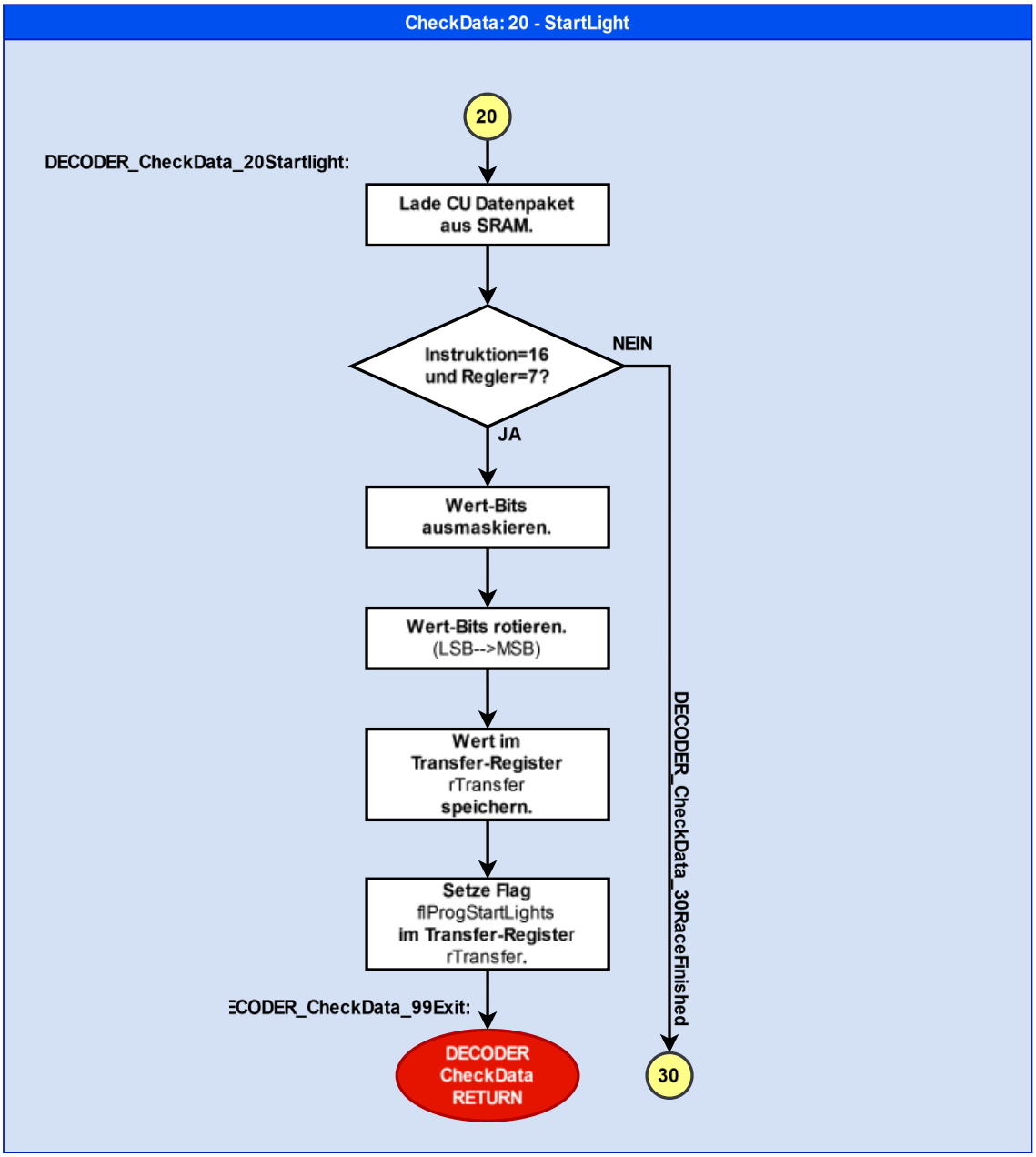
DECODER: SaveData – Empfangene Daten im SRAM speichern



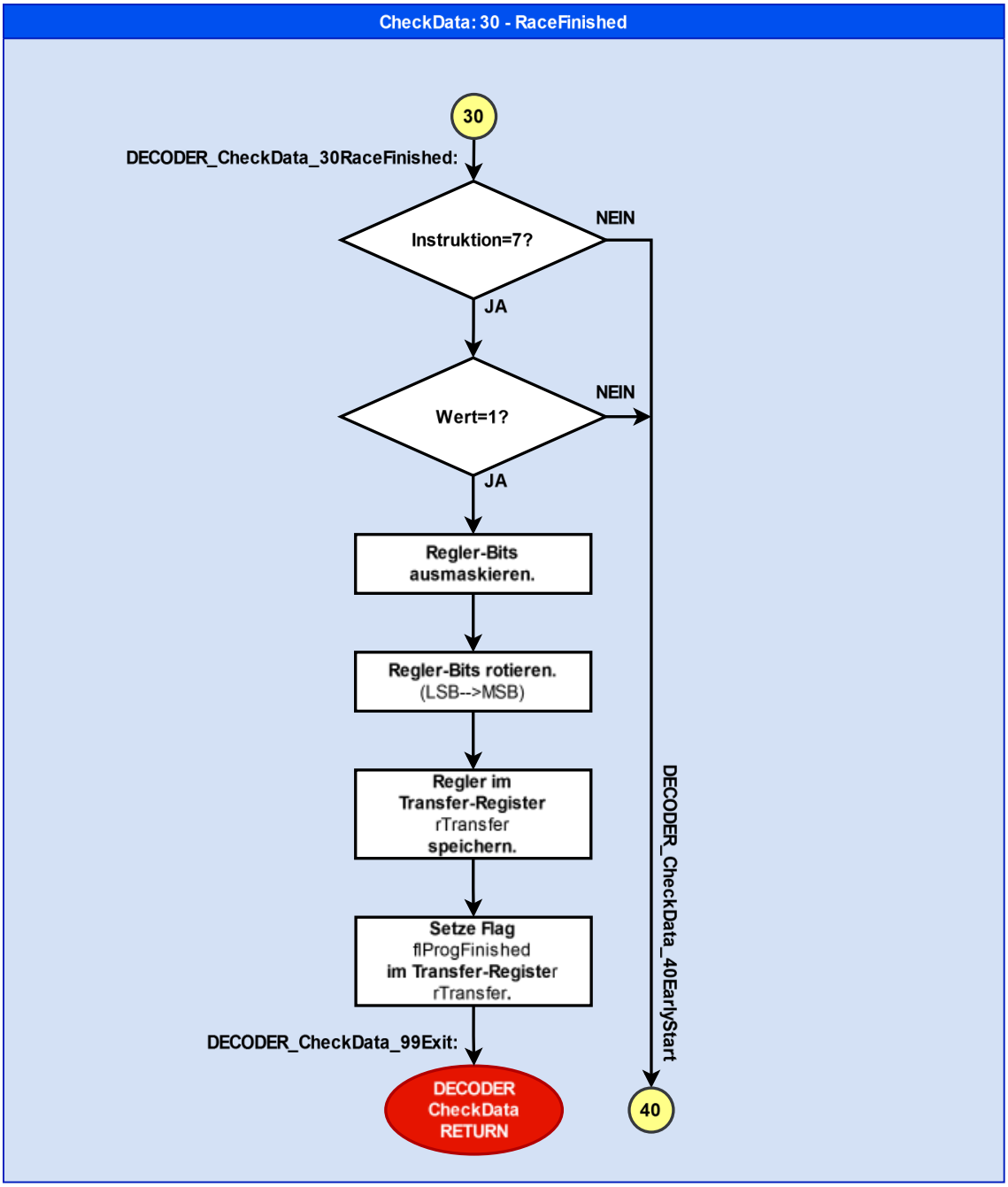
DECODER: CheckData – Teil 10 – Reset



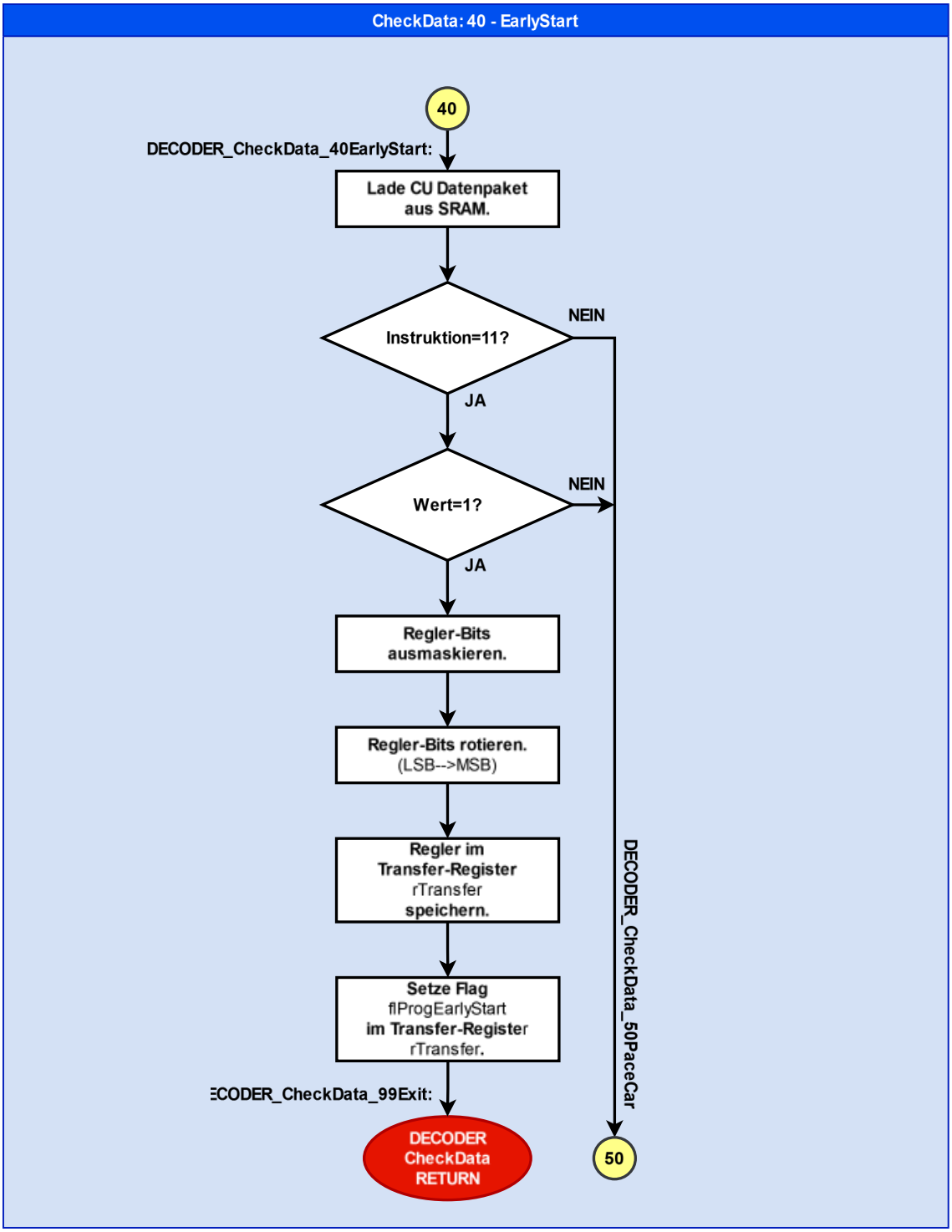
DECODER: CheckData – Teil 20 – Start-Sequenz



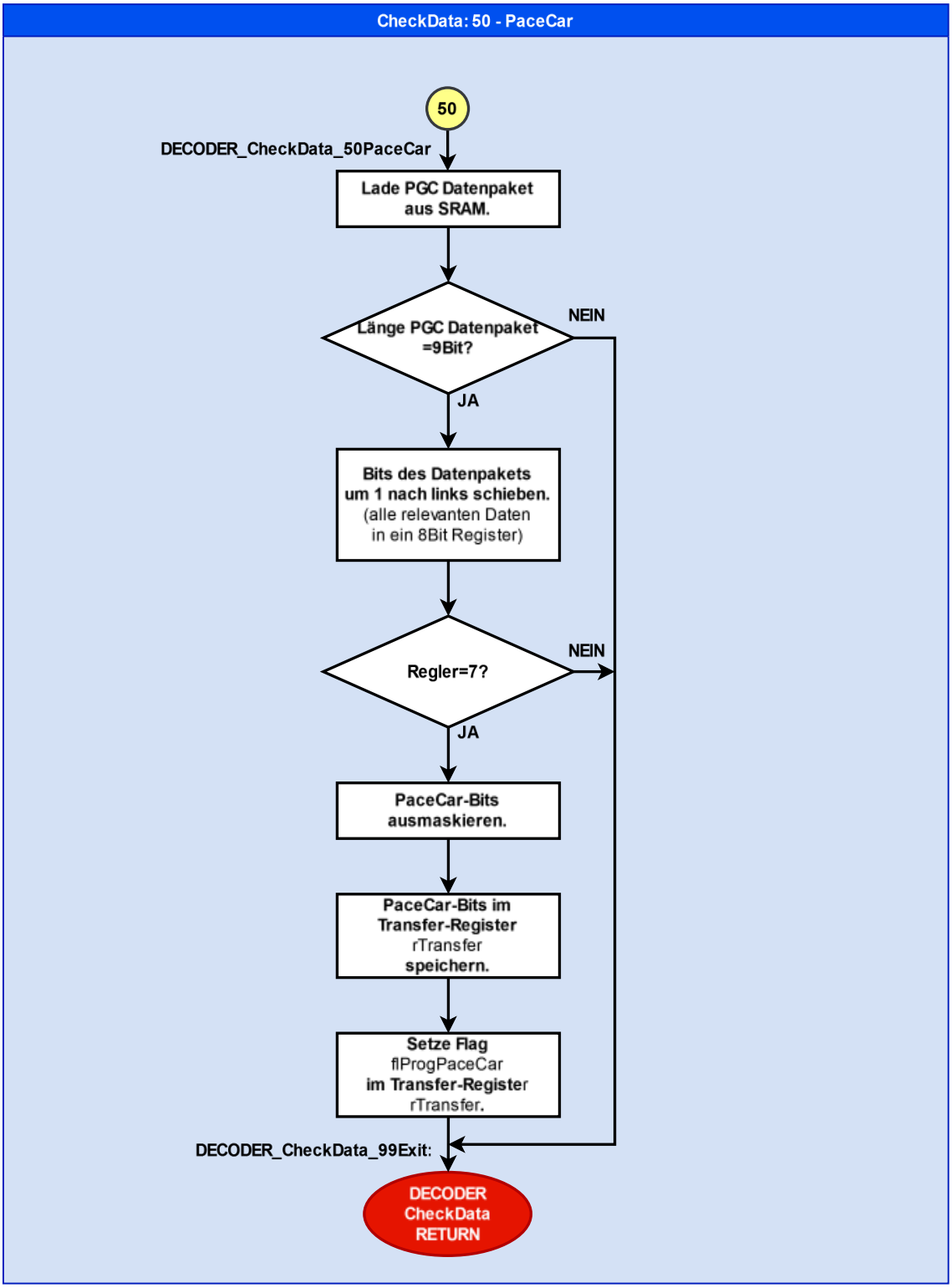
DECODER: CheckData – Teil 30 – Rennen beendet



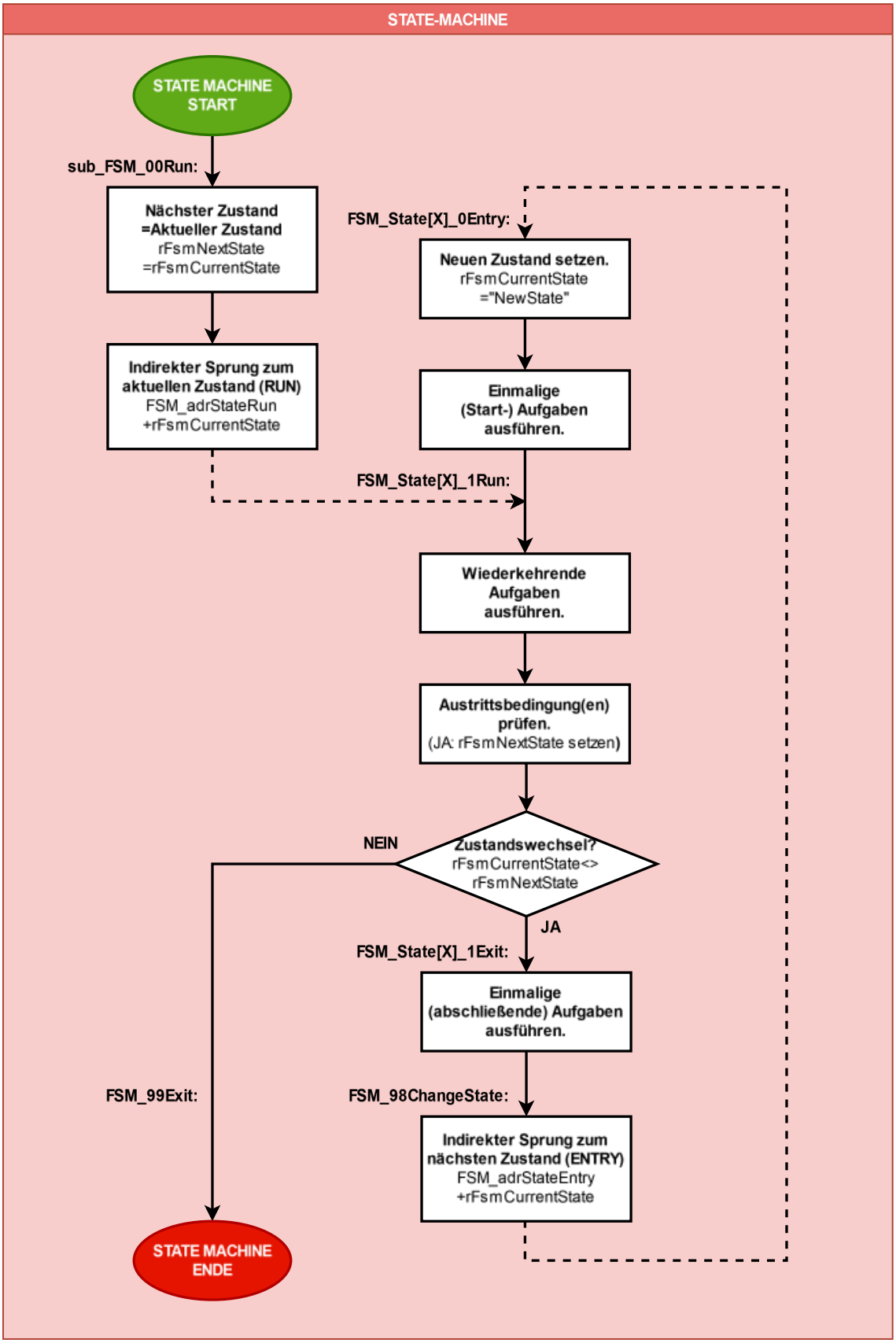
DECODER: CheckData – Teil 40 – Frühstart



DECODER: CheckData – Teil 50 – Pace Car (Safety-Car)



STATE MACHINE: Grundsätzlicher Aufbau



Querschnittliche Konzepte

Quellcode-Aufteilung in Dateien

Assembler-Dateien (.asm) für jede logische Funktionalität

Um den Assemblercode möglichst übersichtlich zu strukturieren, wird der Quellcode in logische Funktionsblöcke – Module – aufgeteilt (hier: main, decoder und state machine). Für jeden Funktionsblock bzw. jedes Modul wird grundsätzlich eine eigene Assembler-Datei im Projekt erstellt. So können diese Module auch in anderen Projekten wieder verwendet oder einfach durch neue ausgetauscht werden.

Das Hauptprogramm erhält hierbei den Dateinamen „main.asm“. Die Assembler-Dateien (.asm) der Module werden am Ende der main.asm per include-Direktive eingebunden

Include-Dateien (.inc)

Für jede Assembler-Datei (.asm) wird eine eigene Include-Datei (.inc) mit gleicher Bezeichnung angelegt.

In die Include-Dateien werden Deklarationen, wie beispielsweise Festlegungen und Konfigurationen von Registern und I/O-Ports, sowie Makros und ähnliches, ausgelagert.

Hier gilt: Alles, was beim Kompilieren ohne Weiteres keinen Programmcode erzeugt, kommt in die Include-Datei. Alles andere bleibt in der Assembler-Datei (siehe oben).

Damit die Deklarationen bereits beim Kompilieren des Quellcodes zur Verfügung stehen, werden die Include-Dateien jeweils am Anfang der main.asm per include-Direktive eingebunden.

Bezeichnung von Registern, Variablen, etc.

Präfix

Um bei Bezeichnungen von Registern (.def), I/O-Ports, etc. den Verwendungszweck erkennen zu können, sind möglichst folgende Präfixe in Kleinbuchstaben in Anlehnung zur „ungarischen Notation[21]“ zu verwenden:

Präfix	Bedeutung	Präfix	Bedeutung
cfg…	Konfigurationsparameter	mac…	Makro
fl…	Flag/Bit	msk…	Bitmaske
io…	I/O-Port	p…	Pointer/Zeiger
isr…	Interrupt Service Routine	r…	Register

Schreibweise

Im Anschluss des Präfixes folgt die Bezeichnung in Camel-Case[22]-Schreibweise.

Beispiel:

```
flTransmComplete
```

[21] Ungarische Notation: https://de.wikipedia.org/wiki/Ungarische_Notation
[22] Camel-Case: https://de.wikipedia.org/wiki/Binnenmajuskel#Programmiersprachen

Bezeichnung von Sprungmarken

Präfix - Module

Grundsätzlich wird einer Sprungmarke der Name des logischen Funktionsblocks bzw. Moduls oder der Bezeichnung der Assembler-Datei (.asm), in der sich die Sprungmarke befindet, in Großbuchstaben vorangestellt.

Präfix - Unterprogramme

Sprungmarken, die als Einstiegspunkt in ein Unterprogramm (Subroutine) dienen, also mit einem „call“-Befehl aufgerufen und mit „ret“-Befehl beendet werden und anschließend an den Aufrufpunkt wieder zurückspringen, wird ein „sub“ vorangestellt. So können Einstiegsmarken für Unterprogramme von reinen Sprungmarken innerhalb eines Programmes unterschieden werden.

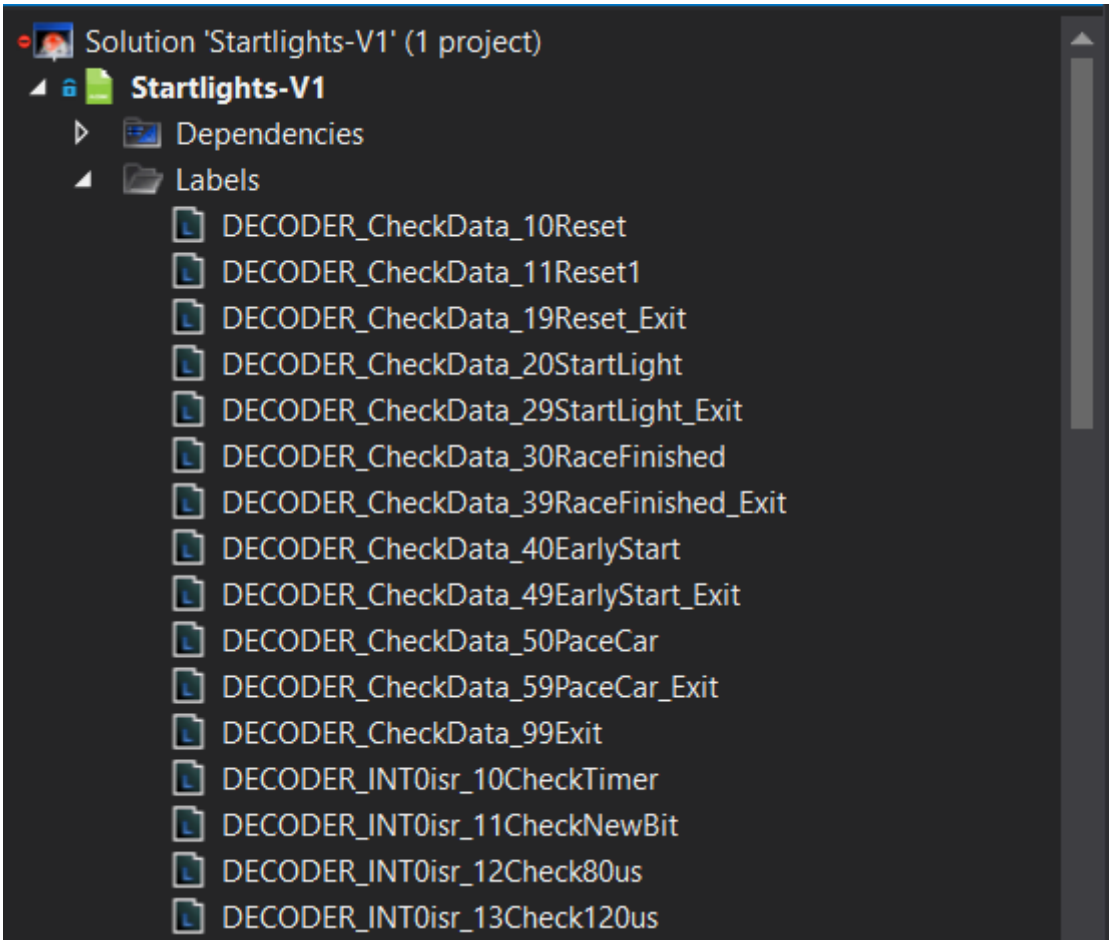
Schreibweise

Im Anschluss des Präfixes folgt die Bezeichnung in Camel Case[23]-Schreibweise.

Nummerierung

Um die Reihenfolge und logische Gruppierung von Sprungmarken zu erreichen sind vor die weitere Bezeichnungen Nummerierungen voranzustellen. Dadurch wird das Auffinden und Zuordnen von Sprungmarken und insbesondere die Navigation im Solution Explorer des Atmel/Microchip-Studios deutlich vereinfacht.

Beispiele:



[23] Camel Case: https://de.wikipedia.org/wiki/Binnenmajuskel#Programmiersprachen

Zustandsautomat (finite state machine

Die Zustände des Zustandsautomaten[24] sind in die drei Teile

- **Einstieg** (Entry)
- **Betrieb** (Run)
- **Ausstieg** (Exit)

zu gliedern.

Siehe: STATE MACHINE: Grundsätzlicher Aufbau

Einstieg (Entry) / Eingangsaktionen

Wird initial in einen neuen Zustand gewechselt, erfolgt der Aufruf des Zustands stets einmalig am Teil Einstieg (Entry).

Hier werden alle Aufgaben ausgeführt, die einmalig beim Erreichen des jeweiligen Zustands auszuführen sind, wie beispielsweise die Zustandsanzeige einschalten.

Anschließend wird vom Einstieg in den Betrieb (Run) übergeleitet.

Betrieb (Run) / Eingabeaktionen

Wird ein Zustand – ohne Zustandswechsel – erneut aufgerufen, erfolgt dies stets im Teil Betrieb (Run).

Hier werden alle Aufgaben ausgeführt, die innerhalb des Zustandes wiederholt auszuführen sind.

Dies sind unter anderem die Prüfung von Austrittsbedingungen (Zustandswechseln) und das Einleiten von Zustandswechseln.

Hat ein Zustandswechsel zu erfolgen, wird der neue Zustand festgelegt und in den Teil Ausstieg (Exit) gesprungen.

Bleibt der Zustand erhalten, wird – je nach Programmaufbau – der Zustandsautomat bis zum nächsten Aufruf verlassen oder wieder an den Anfang des Programmteils Betrieb (Run) des aktuellen Status gesprungen.

Ausstieg (Exit) / Ausgangsaktionen

Der Ausstieg aus einem Zustand beim Zustandswechsel hat immer über den Teil Ausstieg (Exit) zu erfolgen.

Hier sind alle abschließenden Aufgaben durchzuführen, wie beispielsweise die Zustandsanzeige ausschalten.

[24] https://de.wikipedia.org/wiki/Endlicher_Automat

Entwurfsentscheidungen

Decoder

Methode

Zur Dekodierung der Carrera® digital 124/132 Manchester-Datenpakete wurde entschieden, das zeitbasierte Verfahren (Timing Based Manchester Decode) zu verwenden.

Vor dem Hintergrund, dass Datenpakete unterschiedliche Längen haben können und eine möglichst schnelle Auswertung nach Paket-Ende (Frame-Ende) erfolgen soll, wurde entschieden einen Timer-Overflow-Interrupt als Zähler der vergangenen Zeit in 10 µs Schritten und einen externen Interrupt zur Bearbeitung der Flankenwechsel zu verwenden, die unabhängig voneinander arbeiten.

Die Zeitabstandsprüfungen für die Flankenwechsel finden im externen Interrupt mithilfe der Abfrage des Zählers aus dem Timer-Overflow-Interrupt (>80 us und <120 us) statt.

Innerhalb des Timer-Overflow Interrupts erfolgt die Prüfung des Übertragungsendes (>125 us).

Im Beispiel der Atmel Manchester Coding Basics[25] erfolgt die Dekodierung mittels flankengetriggertem Timer. Dies würde allerdings eine jeweils bekannte Länge der jeweiligen Datenpakete voraussetzen, um zeitnah das Paket-Ende (Frame-Ende) zu erkennen. Mit einer Logik könnte zwar nach Empfang des Control-Unit-Datenpaketes die Reihenfolge und damit die Länge der einzelnen Datenpakete der Reihenfolge nach abgearbeitet werden, die oben beschriebene Umsetzung schien hier allerdings einfacher und flexibler.

Zustandsautomat (finite state machine)

Allgemein

Die Steuerung der STARTAMPEL „Bären(n)keller“ im Rahmen eines Zustandsautomaten[26] (finite state machine) ist hier obligatorisch.

Nach jedem Aufruf des Zustandsautomaten wird dieser wieder beendet und springt in das Hauptprogramm zurück. So hat das Hauptprogramm die vollständige Ablaufkontrolle und könnte bei zukünftigem Bedarf für weitere Anwendungen noch zusätzliche Zustandsautomaten bedienen.

CHAOS-Verarbeitung

Grundsätzlich werden die Wechselbedingungen über das Transfer-Register (rTransfer) als Ergebnis der Prüfroutine im Decoder übergeben.

Für die Entscheidung zwischen Start-Vorbereitung und CHAOS muss allerdings ein Merker (rFsmRacing) für den Rennbetrieb verwendet werden. Hierzu wird beim Beginn der Start-Sequenz (erste rote LED-Spalte ein) ein Merker-Register mit 00000001b gesetzt. Bei jedem weiteren Countdown wird das Register um eine Stelle nach links geschoben. Ein Reset (0-Init) oder Ende des Rennens[27] (9-RennEnde) setzen den Merker wieder zurück.

Werden nun während einer freigegebenen Rennstrecke (6-Grün) durch Drücken der Start-Taste an der CARRERA® Control-Unit alle roten LEDs angeschaltet, während das 5. Bit linkseitig des Merker-Registers gesetzt ist (00010000b), wird in den Zustand CHAOS gewechselt.

---

[25] Atmel Manchester Coding Basics: https://ww1.microchip.com/downloads/en/Appnotes/Atmel-9164-Manchester-Coding-Basics_Application-Note.pdf
[26] Zustandsautomat: https://de.wikipedia.org/wiki/Endlicher_Automat
[27] Nur mit CARRERA® Lap-Counter (https://carrera-toys.com/product/20030355-lap-counter) o. ä.

Risiken & technische Schulden

Decoder

Flankenerkennung

- Mit der Startflanke beginnend (Framestart) werden alle nachfolgenden Flankenwechsel im Abstand von 80 µs bis 120 µs als gültige Daten interpretiert.
- Frühere oder spätere Flankenwechsel werden verworfen.
- Erfolgt nach 125 µs kein erneuter Flankenwechsel (Frameende) werden die seit der Startflanke empfangenen Daten zu einem Datenpaket zusammengefasst.

Daraus könnten sich gegebenenfalls folgende Auswertefehler ergeben:

- **Dekodierung beginnt mitten im Übertragungsframe.**
  Wird die Dekodierung bereits während einer laufenden Übertragung gestartet, wird nur ein Teil der Informationen aufgenommen und das Datenpaket ist unvollständig.

- **Flankenwechsel außerhalb des „Zeitfensters“**
  Außerhalb des Zeitfenster zwischen 80 µs und 120 µs empfangene Daten werden ignoriert. Die Dekodierung wird nicht mit Fehler gestoppt. Es findet auch keine Auswertung einer Fehlerrate statt.
  Unerwartete Flankenwechsel könnten in Ausnahmefällen zu Fehlauswertungen führen.

- **Keine Parität oder Prüfsumme**
  Da bei der Carrera® digital 124/132 im Datenprotokoll weder Parität noch Prüfsumme zur Fehlererkennung angefügt werden, lassen sich Übertragungsfehler nicht erkennen.

Folgende Maßnahmen wurden getroffen, um fehlerhafte Datenpakete nicht in der Auswertung zu berücksichtigen:

- **Mindestlänge der Datenpakete**
  Datenpakete benötigen eine Mindestlänge von 7 Bit.
  Kleinere Datenpakete werden verworfen.

- **Prüfung der Datenpaketlänge vor Verarbeitung**
  Für jedes Datenpaket wird die Länge in Bit zusätzlich gespeichert.
  Somit kann vor Verarbeitung geprüft werden, ob das jeweilige Datenpaket auch die korrekte Länge besitzt.

- **Zustandsautomat verhindert unzulässige Zustandswechsel**
  Unerlaubte Zustandswechsel werden durch festgelegte Austrittsbedingungen des Zustandsautomaten (finite state machine) verhindert.

Datenspeicherung/Pufferung

Jedes Datenpaket wird im SRAM an seine vorgesehene Speicheradresse zur Auswertung abgelegt. Ein Neuempfang überschreibt das jeweilige Datenpaket im SRAM. Eine Pufferung bspw. in Form eines Ringspeichers wurde aufgrund des Programmieraufwandes nicht vorgesehen.

Dies bedeutet, dass die Auswertung eines Datenpakets spätestens nach 75 ms abgeschlossen sein muss, da nach dieser Zeit das nachfolgende Datenpaket empfangen und der SRAM überschrieben wird.

Gemäß aktueller Randbedingungen wird der eingesetzte Mikrocontroller mit 8 MHz ohne Vorteiler (CLKDIV) getaktet. Ein Takt benötigt dauert damit 125 ns. Zwischen einzelnen Datenpaketen stehen demnach 6.000 Taktoperationen zur Verfügung. Jeweilige Datenpakete im SRAM werden nach 60.000 Takten überschrieben. Für die aktuell implementierten Funktionen der STARTAMPEL „Bären(n)keller“ ist dies mehr als ausreichend.

Bei Übertragung der Software auf einen anderen Mikrocontroller mit niedrigerer Taktfrequenz oder bei deutlicher Erweiterung des Funktionsumfangs, sowie bei Umsetzung der Software in einer Hochsprache (C, Sketch, etc.) ist sicherzustellen, dass eine Auswertung rechtzeitig abgeschlossen ist.

Glossar

arc42	Frei verfügbares (Open Source) Konzept für die Entwicklung, Dokumentation und zur Kommunikation von Softwareprojekten. https://arc42.org/
Assembler	Maschinenorientierte Programmiersprache https://de.wikipedia.org/wiki/Assemblersprache
AVR® Mikrocontroller	8 Bit-Mikrocontroller-Familie des US-amerikanischen Herstellers Microchip, vormals Atmel. https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus https://de.wikipedia.org/wiki/Microchip_AVR
Control-Unit	Zentraleinheit zur Bedienung, Steuerung und Stromversorgung der CARRERAR® digital 124/132 Modellrennbahn. CARRERA® Control-Unit (Art.-Nr. 20030352) https://carrera-toys.com/de/product/20030352-control-unit
Datenlogger	Zeichnet Daten fortlaufend auf und speichert diese für eine spätere Auswertung ab. https://de.wikipedia.org/wiki/Datenlogger
debugWire	Serielles Übertragungsprotokoll von Atmel (Microchip) für die Programmierung und Fehlersuche bei AVR® Microcontroller. https://de.wikipedia.org/wiki/DebugWIRE
ISP	Schnittstelle zur Programmierung von AVR® Microcontroller (In-System Programming)
Lap-Counter	Renncomputer zur Zeit- und Rundenmessung und Steuerung von Rennen als Ergänzung zur CARRERA® Control-Unit. CARRERA® Lap-Counter (Art.-Nr. 20030355) https://carrera-toys.com/de/product/20030355-lap-counter
Oszilloskop	Elektronisches Messgerät, welche den zeitlichen Verlauf von elektrischen Spannungen optisch darstellen kann. https://de.wikipedia.org/wiki/Oszilloskop
PACE CAR (SAFETY CAR)	Fahrzeug zur Absicherung bei Gefahrensituationen im Motorsport. https://de.wikipedia.org/wiki/Safety-Car
Sketch	Entwicklungsumgebung einer C-ähnlichen Programmiersprache für die Arduino Microcontroller-Entwicklungsplattform. https://www.arduino.cc/ https://de.wikipedia.org/wiki/Arduino_(Plattform)
SRAM	Static random-access memory https://de.wikipedia.org/wiki/Static_random-access_memory Arbeitsspeicher des Mikrocontrollers
Startampel Startlight	Startampel zur Signalisierung der Startfreigabe für den stehenden Start im Motorsport. https://de.wikipedia.org/wiki/Start_(Motorsport) CARRERA® Startlights (Art.-Nr. 20030354) https://carrera-toys.com/product/20030354-startlight
Startlight „Bären(n)keller“	Eigenentwicklung einer Startampel mit erweitertem Funktionsumfang. (Start-Sequenz, Früh-Start, Pace-Car, Chaos, Sieger,…) Hier im Dokument beschrieben.
State Machine / Zustandsautomat	Softwaremodell eines Automaten, welcher zum gleichen Zeitpunkt nur einen fest definierten Zustand einnehmen kann. Die Wechsel von einem Zustand in den nächsten sind klar definiert. https://de.wikipedia.org/wiki/Endlicher_Automat

Ergänzende Informationen

Carrera Toys GmbH

▷ **CARRERA Toys GmbH**
https://carrera-toys.com/

▷ **CARRERA® digital 132**
https://carrera-toys.com/digital-132

▷ **CARRERA® digital 124**
https://carrera-toys.com/digital-124

▷ **CARRERA® Control-Unit (20030352)**
https://carrera-toys.com/product/20030352-control-unit

▷ **CARRERA® Handregler/Controller (20030340)**
https://carrera-toys.com/product/20030340-digital-132-124-handregler

▷ **CARRERA® Startlight (20030354)**
https://carrera-toys.com/product/20030354-startlight

▷ **CARRERA® Lap-Counter (20030355)**
https://carrera-toys.com/product/20030355-lap-counter

▷ **CARRERA® Position Tower (20030357)**
https://carrera-toys.com/product/20030357-position-tower

▷ **CARRERA® Adapter Unit (20030360)**
https://carrera-toys.com/de/product/20030360-adapter-unit

Entwickler-Tools

▷ **KiCAD EDA Suite**
Schaltplan und Platinenlayout
https://www.kicad.org/

▷ **Analog Devices LTspice**
Simulation von elektronischen Schaltungen
https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html

▷ **Microchip Studio**
Software-Entwicklung für Mikrocontroller ATmega/ATtiny in Assembler und C
https://www.microchip.com/en-us/tools-resources/develop/microchip-studio

▷ **FreeCAD**
Parametrische 3D-Konstruktionen
https://www.freecadweb.org/

▷ **Ultimaker Cura**
3D printing Software / Slicer
https://ultimaker.com/software/ultimaker-cura/

Ähnliche Projekte

▷ **Dipl. Ing. Peter Niehues**
http://www.wasserstoffe.de/carrera-hacks/

▷ **Stephan Heß**
http://slotbaer.de/carrera-digital-124-132.html

Entwickler und Herausgeber

andreas wahl

Informationstechnik – Kommunikationstechnik – Elektronik
innovativ + kompetent + effizient

https://www.andreas-wahl.de/

STARTAMPEL „Bären(n)Keller“ auf GitHub:

▷ Software
https://github.com/Andreas-Wahl/Startlight-SW

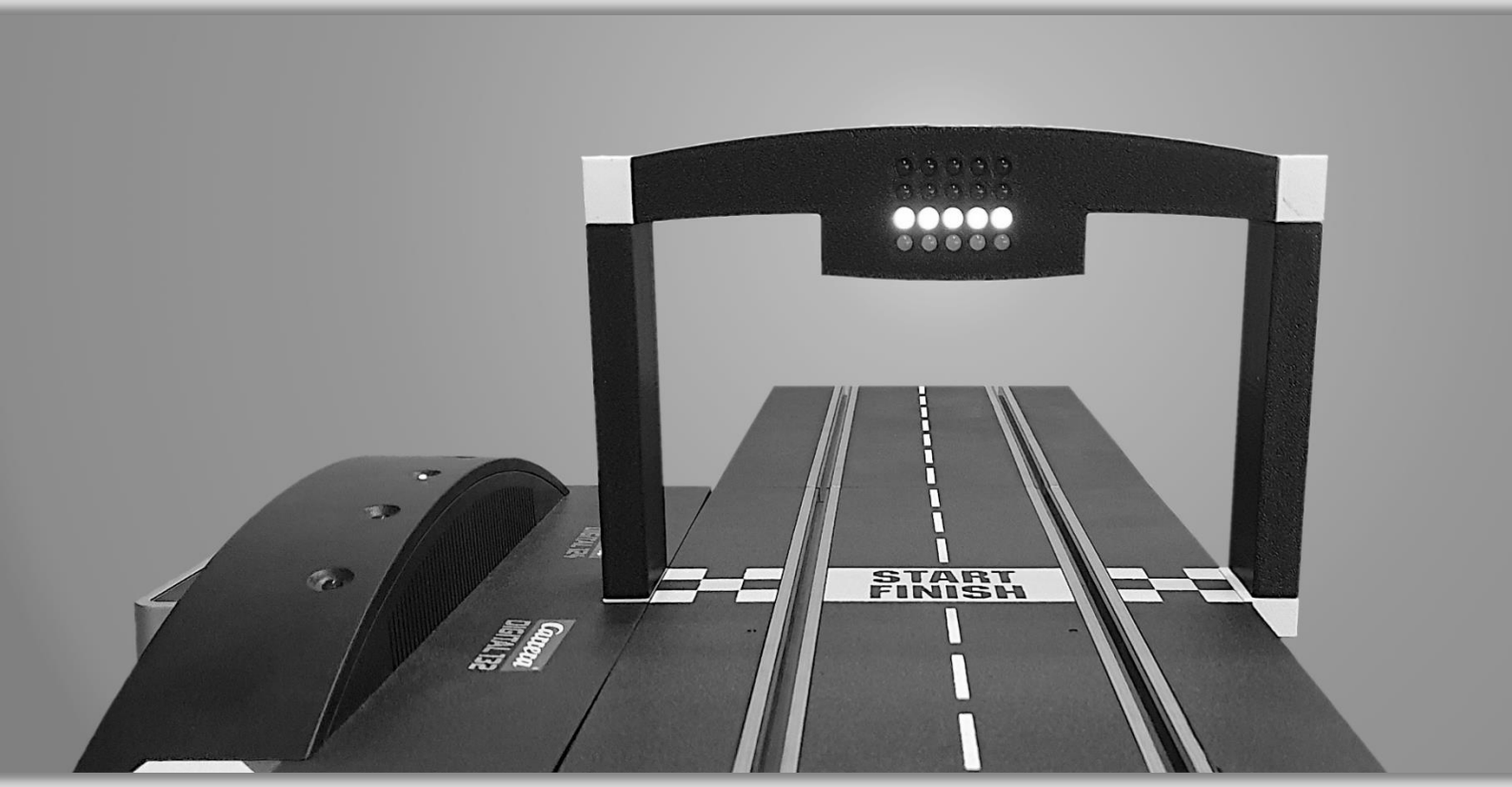
▷ Hardware
https://github.com/Andreas-Wahl/Startlight-HW

▷ Gehäuse
https://github.com/Andreas-Wahl/Startlight-3D

STARTAMPEL – Bären(n)keller auf Thingiverse:

▷ Gehäuse
https://www.thingiverse.com/thing:6443510

▷ Stecker
https://www.thingiverse.com/thing:6444956



andreas wahl

Informationstechnik – Kommunikationstechnik – Elektronik
innovativ + kompetent + effizient