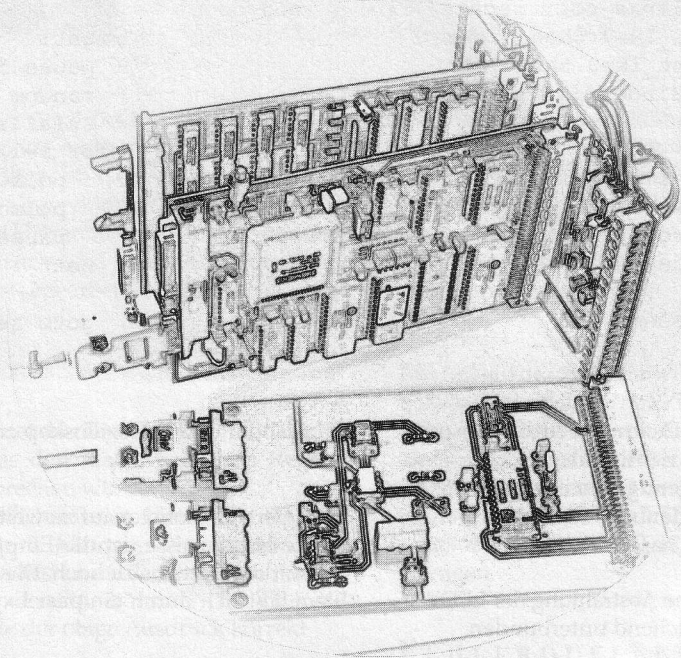


Der CAN-Bus

Intelligente, dezentrale Datenkommunikation für den Praktiker (Teil 4)

Ein CAN-Bus-Interface alleine macht noch kein CAN-Bus-System. Das entsteht erst, wenn das Interface die Verbindung zwischen einem Mikrocontroller oder einem PC und dem CAN-Bus tatsächlich herstellt. Nach der Beschreibung der Hardware im letzten Heft steht jetzt die Softwareseite der Interface-Ansteuerung im Vordergrund.



Zur Realisierung eines CAN-Bus-Systems benötigt man pro CAN-Knoten neben dem CAN-Bus-Interface zumindest einen Mikrocontroller mit der entsprechenden Software. Für die Inbetriebnahme, den Test und den endgültigen Betrieb des Systems sind zwei Arten von Softwarepaketen notwendig: Eine **Betriebssoftware** und eine **Anwendungssoftware** (Applikationssoftware).

Die Betriebssoftware dient zum Ajustieren der CAN-Bus-Interfaces mit dem jeweiligen, nachgeschalteten Mikrocontroller-System. Hiermit kann man feststellen, ob die Ansteuerung vom Mikrocontroller aus einwandfrei funktioniert, ob das CAN-Bus-Interface hard- und softwaremäßig korrekt arbeitet und ob die Daten entsprechend über den Bus transportiert werden. Es läßt sich so eine einfache Kommunikation zwischen zwei bzw. mehreren Busteilnehmern aufbauen. Die

Applikationssoftware hängt von der jeweiligen Aufgabenstellung für den CAN-Bus ab, z.B. Meßwerterfassung, Ansteuerung von Displays, Transfer von Uhrzeit und Daten und so weiter. Jede Busstation enthält dann ihr individuelles Softwarepaket für die betreffende Aufgabe. Die Summe aller Teilfunktionen, die auf jeder Station ablaufen, ergibt dann die gewünschte Gesamtfunktion des Feldbussystems. Mit anderen Worten: als Endergebnis läßt sich das räumlich verteilte System steuern, regeln und überwachen.

DIE BETRIEBS SOFTWARE

Bei der Programmerstellung für den SJA1000 gelten die gleichen allgemeinen Grundsätze wie bei der Programmierung anderer externer Peripherie-Einheiten:

1. Die Funktion des SJA1000 wird

CAN ADDRESS	SEGMENT	OPERATING MODE		RESET MODE	
		READ	WRITE	READ	WRITE
0	control	control	control	control	control
1		(FFH)	command	(FFH)	command
2		status	-	status	-
3		interrupt	-	interrupt	-
4		(FFH)	-	acceptance code	acceptance code
5		(FFH)	-	acceptance mask	acceptance mask
6		(FFH)	-	bus timing 0	bus timing 0
7		(FFH)	-	bus timing 1	bus timing 1
8		(FFH)	-	output control	output control
9		test	test; note 2	test	test; note 2
10	transmit buffer	identifier (10 to 3)	identifier (10 to 3)	(FFH)	-
11		identifier (2 to 0), RTR and DLC	identifier (2 to 0), RTR and DLC	(FFH)	-
12		data byte 1	data byte 1	(FFH)	-
13		data byte 2	data byte 2	(FFH)	-
14		data byte 3	data byte 3	(FFH)	-
15		data byte 4	data byte 4	(FFH)	-
16		data byte 5	data byte 5	(FFH)	-
17		data byte 6	data byte 6	(FFH)	-
18		data byte 7	data byte 7	(FFH)	-
19		data byte 8	data byte 8	(FFH)	-
20	receive buffer	identifier (10 to 3)	identifier (10 to 3)	identifier (10 to 3)	identifier (10 to 3)
21		identifier (2 to 0), RTR and DLC	identifier (2 to 0), RTR and DLC	identifier (2 to 0), RTR and DLC	identifier (2 to 0), RTR and DLC
22		data byte 1	data byte 1	data byte 1	data byte 1
23		data byte 2	data byte 2	data byte 2	data byte 2
24		data byte 3	data byte 3	data byte 3	data byte 3
25		data byte 4	data byte 4	data byte 4	data byte 4
26		data byte 5	data byte 5	data byte 5	data byte 5
27		data byte 6	data byte 6	data byte 6	data byte 6
28		data byte 7	data byte 7	data byte 7	data byte 7
29		data byte 8	data byte 8	data byte 8	data byte 8
30	(FFH)	-	(FFH)	-	
31	clock divider	clock divider; note 3	clock divider	clock divider	

990066-3-13

Tabelle 1
Die internen SFRs des SJA1000 für den BasicCAN-Modus

durch die Programmierung eines Satzes von internen **Special Function Registern (SFRs)** eingestellt bzw. festgelegt.

- Diese internen SFRs erscheinen für den Mikrocontroller als ganz normale Speicherplätze im externen RAM-Speicherbereich, in die er etwas hineinschreiben bzw. aus denen er etwas herauslesen kann. Der Mikrocontroller weiß also gar nicht, daß er mit einem CAN-Controller zusammenarbeitet. Für ihn bzw. für die Anwendungssoftware sind nur Speicherzugriffe auf bestimmte externe Speicherstellen maßgeblich. Erstellt man daher die Betriebssoftware für den SJA1000, so sind die folgenden Punkte zu klären bzw. zu bearbeiten:
 - ◆ Festlegung der Chip-Select-Basis-

Adresse für den SJA1000.

- ◆ Verständnis des Aufbaus der internen

Struktur der SFRs des SJA1000.

- ◆ Erstellung der Routine zur Grundinitialisierung des SJA1000.
- ◆ Erstellung der Routine zum Aussenden von Daten auf den CAN-Bus.
- ◆ Erstellung der Routine zum Empfang von Daten vom CAN-Bus.

Nachfolgend wollen wir Ihnen die Abarbeitung dieser Punkte für den BasicCAN-Modus des SJA1000 näher vorstellen. Ausführliche und weitergehende Informationen dazu finden Sie im Datenblatt und in den Applikationsschriften zu diesem Baustein [1].

Festlegung der Chip-Select-Basis-Adresse

Unter dieser Adresse wird der Baustein grundsätzlich angesprochen. Da der

SJA1000 im BasicCAN-Modus einen zusammenhängenden externen Adreßbereich von 32 Byte und im PeliCAN-Modus einen Bereich von 128 Byte benötigt, wird hierbei der maximale Bereich von 128 Byte zugrunde gelegt, um einen späteren Betrieb im PeliCAN-Modus nicht auszuschließen. Aktiviert wird der SJA1000 durch ein Low-Signal an seinem CS-Anschluß (Pin 3). Das bedeutet: das Mikrocontroller-System muß seine Adreßdekodierung so aufbauen, daß innerhalb eines zusammenhängenden Adreßbereiches von mindesten 128 Byte Größe solch ein Low-Signal am Pin 8 des Steckers K3 erzeugt wird, um den Datentransfer mit dem CAN-Controller aufnehmen zu können. Die erste Adresse, bei der dieses der Fall ist, ist die sog. **Chip-Select-Basis-Adresse** des SJA1000. Greift der Mikrocontroller nun auf irgendeine Speicherstelle aus diesem Adreßbereich zu, so erhält er

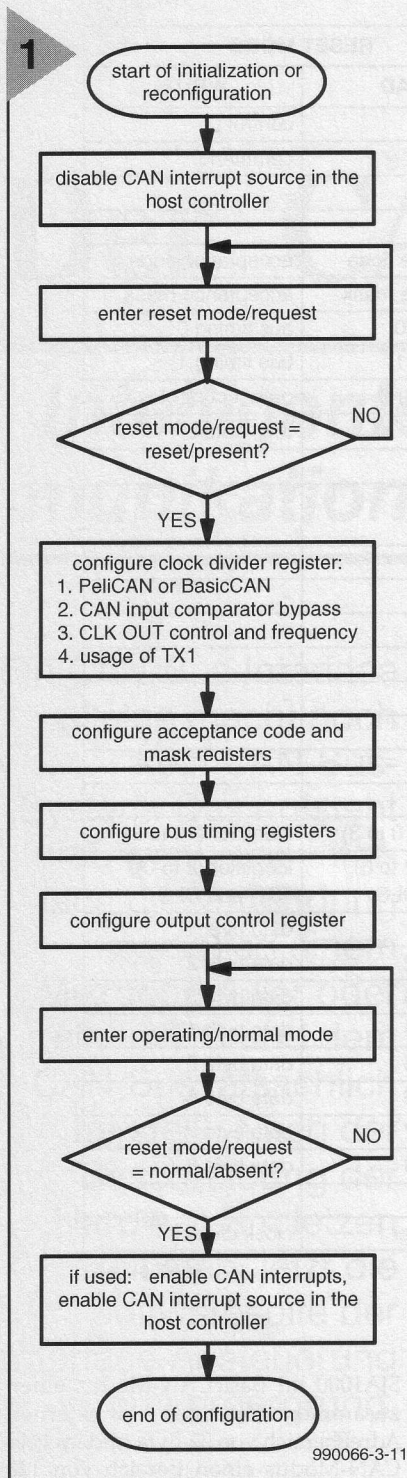


Bild 1. Das Flußdiagramm zur Initialisierung des SJA1000.

den Byte-Inhalt eines SFRs des SJA1000 bzw. so kann er in ein SFR des SJA1000 einen Byte-Wert einschreiben. Wir nehmen für die nachfolgenden Betrachtungen einmal an, die Chip-Select-Basis-Adresse des SJA1000 sei F000h.
Aufbau der internen Struktur der SFRs
 Die für den Betrieb im BasicCAN-Modus maßgeblichen internen SFRs des SJA1000 zeigt die **Tabelle 1**. Es bedeuten hierbei:

- ▶ In der ersten Spalte ("CAN ADDRESS") finden Sie die interne Adresse des jeweiligen SFRs, zu der nun noch die Chip-Select-Basis-Adresse des Bausteins hinzu addiert werden muß. **Beispiel:** Sie möchten auf das Statusregister des SJA1000 zugreifen. Die interne Adresse dieses SFRs ist 2. Dazu wird jetzt noch F000h addiert. Zum Auslesen bzw. zum Beschreiben dieses Registers müssen Sie also in Ihrem Programm einen Zugriff auf die externe RAM-Speicherstelle mit der Adresse F002h programmieren. Das Clock Divider-Register erscheint demnach unter der Adresse F01Fh ($\equiv F000h + 31d$; beachten Sie hier die unterschiedlichen Zahlensysteme!).
- ▶ In der zweiten Spalte sehen Sie die grundsätzliche Einteilung der SFRs in drei verschiedene Gruppen. Es gibt die Kontroll (control)-, die Sende(transmit buffer)- und die Empfangs(receive buffer)-Gruppe.
- ▶ Der SJA1000 unterscheidet zwei softwaremäßig einstellbare Betriebsarten:
 - a) **Operating Mode:** das ist die ganz normale Betriebsart des SJA1000.
 - b) **Reset Mode:** in dieser Betriebsart befindet sich der SJA1000, wenn Sie einen Hardware-Reset auslösen oder das Reset-Bit im Control-Register gesetzt ist. Der SJA1000 stellt dann sofort seinen normalen Betrieb ein. Die Aktivierung dieser Reset-Betriebsart ist dann notwendig, wenn Sie den SJA1000 (Grund)initialisieren wollen, d.h. bestimmte Betriebsparameter lassen sich nur im Reset-Mode einstellen. Dazu wird dann sinnvollerweise zuerst das Reset-Bit gesetzt (der SJA1000 stellt seinen normalen Betrieb ein), die gewünschten Parameter werden geändert und abschließend wird das Reset-Bit wieder zurückgesetzt. Danach arbeitet der SJA1000 mit den neuen Parameterwerten weiter.
- ▶ Die Spalten drei und vier zeigen:
 - die Funktionen der Register,
 - die Bedeutung des Inhaltes beim Auslesen der Register
 - die Bedeutung des Inhaltes beim Einschreiben in die Register im Operating Mode.
- ▶ In den Spalten fünf und sechs stehen die entsprechenden Angaben für die Register im Reset Mode. Als **Beispiel** ein internes SFR mit der Adresse 4:
 - **Operating Mode** ("Normaler Betrieb des SJA1000"):
 - Read: Ein Auslesen dieses Registers ist zwar möglich, bringt aber keine sinnvoll verwertbaren Ergebnisse, denn der Auslesewert ist immer FFh.

- Write: Ein Beschreiben dieses Registers ist nicht möglich.
- **Reset Mode** ("SJA1000 befindet sich im Reset-Zustand"):
 - Read: Ein Auslesen dieses Registers ergibt den Wert des Akzeptanz-Codes.
 - Write: Durch Einschreiben in dieses Register kann ein neuer Akzeptanz-Code gesetzt werden.

Mit anderen Worten: dieses interne SFR hat im normalen Betrieb des SJA1000 keine besonderen Funktionen. Im Reset-Mode wird jedoch hierüber der Akzeptanz-Code festgelegt, mit dem der SJA1000 dann im normalen Betrieb arbeitet.

Erstellung der Routine zur Grundinitialisierung

Bei der Erstellung dieser Routine hilft ein "scharfer Blick" in die Unterlagen zum SJA1000 weiter, hier in die Application Note AN97076, S. 23ff, [1]. Dort gibt der Hersteller bereits ein sehr ausführlich kommentiertes Flußdiagramm an, nach dem die Initialisierung des SJA1000 vonstatten gehen sollte (**Bild 1**).

Wenn Sie sich jetzt noch die Beschreibungen zu den einzelnen Registern genauer ansehen, können Sie die Parameter-Werte ihren speziellen Wünschen entsprechend leicht einstellen.

Erstellung der Routine zum Aussenden von Daten

Wie schon mehrfach erwähnt, nimmt der CAN-Controller SJA1000 dem Anwender mehr als 99% der Aufgaben der "Sende-Betriebsabwicklung" ab. Um Byte-Daten auf dem CAN-Bus auszusenden, sind lediglich vier Aktionen notwendig:

- ▶ Übergabe des gewünschten Nachrichten(Objekt)-Identifiers für den auszusendenden Frame an den SJA1000.
- ▶ Angabe, wie viele Datenbytes gesendet werden sollen (0 bis 8 Stück).
- ▶ Festlegung, ob dieser Frame ein RTR(Remote-Transmission Request)-Frame ist oder nicht.
- ▶ Einschreiben der auszusendenden Nutzdatenbytes in den Sendebuffer des SJA1000.

Mehr ist nicht erforderlich! Den Rest führt der CAN-Controller automatisch und selbständig durch, nämlich

- ▶ "Zusammenbau" des Frames
- ▶ Berechnung der CRC-Summe
- ▶ Belegung der anderen Felder im Frame
- ▶ Zugriff auf den Bus (Busarbitrierung)
- ▶ Aussenden des Frames
- ▶ Fehlerüberprüfung und so weiter.

Über das Statusregister erhält der