Handbuch zur Inbetriebnahme des Identifikationssystems

IDENTControl IC-KP-B6

PEPPERL+FUCHS
IDENT

Inhaltsverzeichnis

1. Hardwareaufbau

1.1 Eingesetzte Geräte

In der nachfolgenden Tabelle sind alle Komponenten aufgelistet, die zum Anschluss einer IDENTControl IC-KP-B6-SUBD/-V15B an eine SIMATIC S7-400 Steuerung mit Profibus-Anbindung benötigt werden.

KOMPONENTE	HERSTELLERBEZEICHNUNG
Simatic S-7 400 Spannungsversorgung	PS 407 4A
Simatic S-7 400 CPU	CPU-412-2
IDENTControl	IDENTControl IC-KP-B6-SUBD/-V15B
4 x Schreib-/Leseköpfe	IQH-18GM-V1
4 x Anschlusskabel Schreib-/Leseköpfe	V1-G-0,6M-PUR-V1-W
Profibus Verbindungskabel	-
Datenträger	IQC21-58
1 x Anschlusskabel IDENTControl	V1-G-2M-PUR

Tab. 1: Auflistung der verwendeten Hardwarekomponenten

Diese Geräte sind dem in diesem Handbuch beschriebenen Testaufbau entnommen. Die Inbetriebnahme des Identifikationssystems IDENTControl IC-KP-B6-SUBD/-V15B kann auch mit anderen, funktionsgleichen Steuerungseinheiten durchgeführt werden. Die dazu benötigten Informationen sind aus den zugehörigen Handbüchern der SIMATIC Steuerung zu entnehmen.

1.2 Aufbau und Anschluss

Der Aufbau der einzelnen Komponenten ist schematisch in der nachfolgenden Grafik dargestellt.

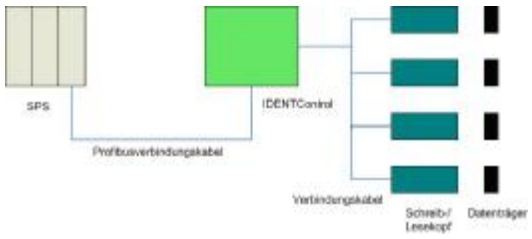

Abb. 1: Aufbau der Hardwarekomponenten

Für die Realisierung des Aufbaus müssen zuerst die Schreib-/Leseköpfe über die Verbindungskabel mit den zugehörigen Anschlussbuchsen der IDENTControl verbunden werden. Hierbei muss beachtet werden, dass aufgrund der gegenseitigen Beeinflussung, der richtige Abstand zwischen zwei Schreib-/Leseköpfen eingehalten wird. Der korrekte Abstand ist dem Datenblatt des jeweiligen Schreib-/Lesekopfes zu entnehmen.

Die Datenträger müssen sich im Erfassungsbereich der Schreib-/Leseköpfe befinden. Bei der Positionierung der Datenträger gilt zu beachten, dass sich immer nur ein Datenträger im Erfassungsbereich eines Schreib-/Lesekopfes befindet.
Anschließend muss die IDENTControl mit einer Versorgungsspannung 20 … 30 V DC über das zugehörige Anschlusskabel verbunden werden.
Im letzten Schritt wird die Kommunikationsverbindung zwischen IDENTControl und der SPS über ein Profibusverbindungskabel hergestellt. Dabei müssen die Kabelenden über einen Leitungsabschlusswiderstand terminiert werden, wenn die IDENTControl als letztes Gerät im Bussegment betrieben wird. Über den genauen Aufbau der Komponenten der SPS wird an dieser Stelle nicht eingegangen. Dazu sei auf das Buch „Automatisierungssystem S7-400 Installationshandbuch“ verwiesen. Weitere Informationen bezüglich der Inbetriebnahme sind aus dem Handbuch „IDENTControl IC-KP-B6-SUBD/-V15B“ (www.pepperl-fuchs.com) zu entnehmen.

1.3 Konfiguration in der Steuerung

In diesem Abschnitt erfolgt die Beschreibung der Konfiguration der Hardware innerhalb der Programmiersoftware „SIMATIC Manager“. Die Konfiguration wird anhand eines Beispiels durchgeführt. Für eine kundenspezifische Konfiguration müssen die Parameter der Einsatzumgebung entsprechend angepasst werden. Die Parameter sind direkt an der IDENTControl über die Bedienfeldtasten einzustellen. Für die Durchführung der Parametereinstellung sei auf das Handbuch „IDENTControl IC-KP-B6-SUBD/-V15B“ (www.pepperl-fuchs.com) verwiesen. Es gilt zu beachten, dass im Auslieferungszustand der IDENTControl bereits Default-Parameter eingestellt sind. Die Default-Einstellung kann ebenfalls dem Handbuch entnommen werden. Nachfolgende Tabelle schlüsselt die im Beispiel eingestellten Parameter auf.

PARAMETER	WERT
Profibus Adresse IDENTControl	3
Übertragungsrate	1,5 MBit/s
Datenträgertyp	„21“ **à** IQC21

Tab. 2: Parameter zur Einstellung der Hardware

Zur Inbetriebnahme der IDENTControl wird zunächst ein neues Projekt gestartet. Nach dem Start des Projektes werden die Komponenten des Steuerungssystems über die Hardware-Konfiguration in das Projekt eingebunden. Im ersten Schritt wird eine SIMATIC400-Station in das Projekt implementiert.

Abb. 2: Einbindung SIMATIC400-Station

Anschließend erfolgt der Aufruf der Hardware-Konfiguration durch Doppelklick auf das Symbol „Hardware“. In der Hardware-Konfiguration werden die benötigten Komponenten Rack, Spannungsversorgung sowie CPU eingefügt. Im nächsten Schritt muss ein DP-Mastersystem eingefügt werden. Über das DP-Mastersystem erfolgt später die Kommunikation. Nachfolgendes Bild zeigt die Oberfläche nach der Zuweisung der Hardware Komponenten.

Abb. 3: Hardware-Konfiguration

Anschließend müssen die Eigenschaften der DP-Kommunikationsschnittstelle der Steuerung festgelegt werden. Dazu muss das Eigenschaftenfenster der DP-Schnittstelle geöffnet werden. Durch Doppelklick auf das zugehörige Symbol im oberen Bild wird das Eigenschaftenfenster geöffnet. In diesem Fenster werden die aktuellen Eigenschaften angezeigt. Um die aktuellen Einstellung zu verändern, muss das Feld „Eigenschaften“ angeklickt werden. Dadurch öffnet sich das folgende Fenster.

Abb. 4:Einstellung DP-Kommunikationsschnittstelle SPS

In diesem Fenster werden die Parameter für die DP-Schnittstelle der SPS eingestellt. Dazu ist die Profibusadresse der SPS anzugeben. Im letzten Schritt muss das Subnetz neu vernetzt werden. Die Eigenschaften des Kommunikationsnetzes sind in einem separaten Fenster einzustellen. Die neu festgelegten Parameter werden über das Feld „OK“ übernommen. Das nachfolgende Bild zeigt das Fenster zur Einstellung der Profibus Kommunikation.

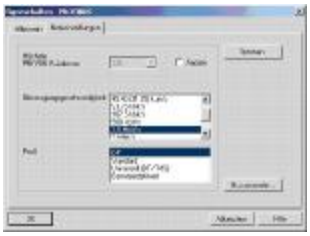

Abb. 5: Einstellung Profibus Kommunikation

Im nächsten Schritt muss die IDENTControl an das Profibusnetz angebunden werden. Dazu wird aus dem Hardware-Katalog das Symbol „IC-KP-B6“ für die IDENTControl in das DP-Mastersystem eingefügt. Anschließend muss die Größe der Datenpakete festgelegt werden. Dazu ist aus dem Hardware-Katalog die entsprechende Einstellung „In/Out X Bytes“ in Verbindungstabelle im unteren Bildabschnitt einzufügen.

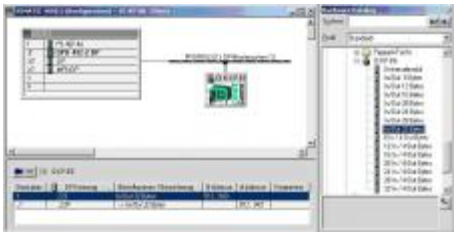

Abb. 6: Einstellung Kommunikationsschnittstelle IDENTControl

In dem Beispiel zur Inbetriebnahme wurde eine Größe der Datenpakete von 32 Bytes festgelegt. Die maximale Größe der Datenpakete ist abhängig von der verwendeten CPU. Genauere Informationen sind aus dem Handbuch der verwendeten CPU zu entnehmen.

Bei der Einstellung der Hardware-Konfiguration ist die Angabe einer Data Hold Time erforderlich. Die Data Hold Time ist ein gerätespezifischer Parameter und gibt die Zeit an, wie lange die Daten im Ausgangsdatenfeld der IDENTControl stehen. Der eingestellte Wert der Data Hold Time ist immer größer zu wählen als die Zykluszeit innerhalb des OB 1.

Abb. 7: Einstellung Data Hold Time

Somit ist die Hardware Anbindung der IDENTControl an die Steuerung beendet und nun erfolgt die Integration des Identifikationssystems auf Basis der Software.

2. Software

2.1 Befehle des Steuerungsprogramms

In diesem Abschnitt wird zunächst die Unterscheidung der verschiedenen Befehlsarten der IDENTControl erläutert, welche mit Hilfe des Steuerungsprogramms durchgeführt werden können. Anschließend erfolgt eine Beschreibung des Ablaufes zur Durchführung eines Befehls. Im letzten Teil wird der strukturelle Aufbau eines Beispielbefehls verdeutlicht.

2.1.1 Befehlsarten

Die Befehle der IDENTControl, die mit Hilfe des Steuerungsprogramms ausgeführt werden können, sind in zwei Arten zu untergliedern. Man unterscheidet zwischen einem Single- und einem Enhanced-Befehl.

Bei einem Single-Befehl erfolgt die Befehlsausführung einmalig, sofern sich ein Code-/Datenträger im Erfassungsbereiches des Schreib-/Lesekopfes befunden hat. Sobald bei der Befehlsausführung der IDENTControl kein Code-/Datenträger erfasst werden konnte, wird die Befehlsausführung durch die Steuerung wiederholt, bis der Befehl vollständig und richtig ausgeführt oder die maximale Anzahl der Wiederholungen überschritten wurde. Die Anzahl der Wiederholungen ist vom Anwender selbst festzulegen. Dies ist notwendig, um eine Blockierung des Schreib-/Lesekopfes durch die dauerhafte durchgeführte Befehlsausführung zu verhindern.

Bei verschiedenen Anwendungen ist aber eine dauerhafte Befehlsausführung erforderlich. Die dauerhafte Befehlsausführung wird über einen Enhanced-Befehl realisiert. Der Befehl wird solange ausgeführt, bis die Bearbeitung durch einen Quit-Befehl abgebrochen wird.

2.1.2 Befehlsablauf

Zunächst werden die für die Ausführung eines Befehls notwendigen Befehlsparameter durch das Steuerungsprogramm an die IDENTControl übertragen. Die IDENTControl sendet bei der Annahme des Befehls die Statusrückmeldung (FF)h an die Steuerung zurück. Durch die Statusrückmeldung wird signalisiert, dass die IDENTControl den Befehl angenommen hat und bearbeitet. Anschließend leitet die IDENTControl den Befehl an die Schreib-/Leseköpfe weiter und wartet auf eine Rückantwort von den Köpfen. Die Rückantwort von den Schreib-/Leseköpfen wird durch die IDENTControl an die Steuerung weitergeleitet. Innerhalb der Steuerung werden die

empfangenen Daten ausgewertet und dem Anwender zu Verfügung gestellt. Bei der Durchführung eines Enhanced-Befehls werden solange die Rückantworten der Schreib-/Leseköpfe durch die IDENTControl an die Steuerung weitergeleitet, bis der Befehl abgebrochen wird.

2.1.3 Befehlsstruktur

Um einen Befehl mit der IDENTControl ausführen zu können, müssen vorher die zugehörigen Befehlsparameter an die IDENTControl übertragen werden. Die Befehlsparameter sind byteweise zusammengefasst und bilden ein Befehlstelegramm. Die Anzahl der zu übertragenden Parameter sowie der Befehlscode sind unterschiedlich, jedoch sind die Strukturen der verschiedenen Befehle identisch. Die nachfolgende Tabelle beschreibt den Aufbau eines Befehlstelegramms für einen Single-Read-Data-Befehl. Dieses Telegramm wird von der Steuerung zur IDENTControl gesendet und bewirkt das Einlesen von Nutzdaten.

BYTE	INHALT / VARIABLE	BITBELEGUNG							
0	#Head_X.OutData.CommandCode	0	0	0	1	0	0	0	0
1	#Head_X.OutData.Channel	Datenwortanzahl				Kanalnummer			T
2	#Head_X.OutData.Wortadr_High	Anfangsadresse high Byte							
3	#Head_X.OutData.Wortadr_Low	Anfangsadresse low Byte							
4..(31)63	#Head_X.OutData.DW1 ... DW(7)15	unbenutzt							

Tab. 3: Aufbau Befehlstelegramm Single-Read-Data Befehl

Der Befehlsparameter #Head_X.OutData.CommandCode enthält den Befehlscode des zu übertragenden Befehls. Für ein Single-Read-Data Befehl hat der Befehlscode die binäre Codierung (10000)b. Eine ausführliche Auflistung der verschiedenen Befehlscodes für die IDENTControl ist dem Handbuch „IDENTControl IC-KP-B6-SUBD/-V15B“ (www.pepperl-fuchs.com) oder der Befehlsliste im Anhang zu entnehmen.

Mit Hilfe des Befehlsparameters #Head_X.OutData.Channel werden zwei Befehlseigenschaften übertragen. Durch die oberen Bits wird die Anzahl der zu lesenden/schreibenden Datenblöcke übertragen. Die Anzahl ist in der Variablen #Head_X.WordNum enthalten und wird während der Befehlszuweisung in den Befehlsparameter übertragen. Die Datenblöcke weisen ein Doppelwort Format auf und haben somit eine Größe von 4 Byte. Die maximale Anzahl der zu übertragenden Datenblöcke ist abhängig von der in der Hardware-Konfiguration getroffenen Einstellung „In/Out X Bytes“. Bei der Einstellung „In/Out 64 Bytes“ können maximal 15 Datenblöcke übertragen werden. Bei der im Beispiel gewählten Einstellung „In/Out 32 Bytes“ werden 7 Datenblöcke übertragen.

Die unteren vier Bits enthalten die Information über den Kanal des Schreib-/Lesekopfes, an welchem der Befehl ausgeführt werden soll. Die nachfolgende Tabelle listet die verschiedenen Kanalcodierungen auf, über die der zugehörige Schreib-/Lesekopf angesprochen wird.

SCHREIB-/LESEKOPF	CODIERUNG	
	dez	dual
1	2	(0010)
2	4	(0100)
3	6	(0110)
4	8	(1000)

Tab. 4: Kanalcodierung der Schreib-/Leseköpfe

Bei der Codierung des Kanals ist zu beachten, dass das niedrigste Bit (LSB) nicht verwendet wird. Bei der Kommunikation über Profibus wird es durch toggeln dazu genutzt, um bei nacheinander folgenden identischen Befehlen eine Unterscheidung treffen zu können.

Mit Hilfe der Parameter #Head_X.OutData.Wordadr_High und #Head_X.OutData.Wordadr_Low kann eine Startadresse festgelegt werden, ab welcher die Datenblöcke gelesen bzw. geschrieben werden. Somit können unterschiedliche Adressbereiche auf dem Datenträger angesprochen werden.

Die Befehlsparameter #Head_X.OutData.DW1..DW15 enthalten die Nutzdatenblöcke. Allerdings bleiben diese Befehlsparameter beim Senden des Befehlstelegramms an die IDENTControl unbenutzt.

Die IDENTControl antwortet auf ein Befehlstelegramm mit einen Antworttelegramm. Der Aufbau eines Antworttelegramms ist in der nachfolgenden Tabelle dargestellt.

BYTE	INHALT / VARIABLE	BITBELEGUNG							
0	#Head_X. InData.CommandCode	0	0	0	1	0	0	0	0
1	#Head_X. InData.Channel	Datenwortanzahl				Kanalnummer			T
2	#Head_X. InData. Status	Status							
3	#Head_X. InData. ExecutionCounter	Ereigniszähler							
4..(31)63	#Head_X. InData.DW1 ... .DW(7)15	00 .. FF							

Tab 5: Aufbau Antworttelegramm Single-Read-Data Befehl

In dem Antworttelegramm werden die ersten zwei Byte des zuvor gesendeten Befehlstelegramms zurückgesendet. Außerdem enthält das Antworttelegramm eine Statusmeldung, die Aufschluss über den Ausführungszustand des Befehls gibt. Mit Hilfe des Ereigniszählers kann die Anzahl der Befehlsereignisse ermittelt werden. Bei jedem Befehlsereignis wird der Zähler um eins erhöht. Ein Befehlsereignis ist beispielsweise ein Statuswechsel von (00)h auf (FF)h.

Die weiteren Parameter des Antworttelegramms enthalten die eingelesenen Nutzdaten. Die Anzahl der eingelesenen Nutzdaten ist dabei wiederum abhängig von den in der Hardware-Konfiguration getroffenen Einstellungen.

2.1.4 Durchführung der Initialisierung

Zu Beginn der Bearbeitung des Funktionsbausteins müssen zunächst die Schreib-/Leseköpfe initialisiert werden. Die Initialisierung wird für jeden Kopf nacheinander durchgeführt. Mit Hilfe der Initialisierung wird festgestellt, an welchem IDENT-Kanal ein Schreib-/Lesekopf angeschlossen ist.

Bei der Initialisierungs-Routine wird ein Change-Tag Befehl an die IDENTControl gesendet. Durch diesen Befehl wird dem Schreib-/Lesekopf mitgeteilt, mit welchem Datenträger er kommuniziert. Anhand der Statusrückmeldung erkennt der Funktionsbaustein, ob ein Schreib-/Lesekopf am entsprechenden Kanal angeschlossen ist.

Wenn die Initialisierung für einen Kopf abgeschlossen ist, wird entweder das Bit #Head_X.ExistTC oder #Head_X.NotExist gesetzt. Das Bit #Head_X.ExistTC ist gesetzt, wenn ein Schreib-/Lesekopf an dem zugehörigen Kanal angeschlossen ist. Andernfalls ist das Bit #Head_X.NotExist gesetzt.

Bei der Durchführung der Initialisierung muss beachtet werden, dass die richtige Datenträgerkennung zugewiesen wird. Die Kennung für den Datenträger wird der Variablen #Head_X.TagType zugewiesen und muss durch den Anwender selbst festgelegt werden. Die Angabe der Datenträgerkennung erfolgt innerhalb des Programms in hexadezimaler Form. Die Kennung der Datenträger ist aus dem Handbuch „IDENTControl IC-KP-B6-SUBD/-V15B“ unter der Beschreibung des Change-Tag Befehls ersichtlich. Es ist zu beachten, dass die Angabe innerhalb des Handbuches im ASCII-Zeichenformat erfolgt. Eine Umrechnung kann mit Hilfe einer, ebenfalls im Handbuch aufgeführten, ASCII-Tabelle durchgeführt werden. Die Datenträgerkennung kann ebenso aus der Auflistung für die Code-/Datenträger im Anhang entnommen werden. Nach erfolgreich durchgeführter Initialisierung aller Köpfe kann ein Befehl durch die IDENTControl durchgeführt werden.

2.1.5 Durchführung eines Single-Befehls

Ein Single-Befehl wird von der IDENTControl nur einmalig ausgeführt. Zur Ausführung des Befehls ist es erforderlich, dass die zugehörigen Befehlsparameter vom Ausgangsdatenfeld des zugehörigen Kopfes innerhalb der SPS zur IDENTControl gesendet werden. Um das Ausgangsdatenfeld an die IDENTControl senden zu kön-

nen, muss es mit den zugehörigen Befehlsparametern und Datenwörtern belegt werden. Auf den Befehlsaufbau wird an dieser Stelle nicht näher eingegangen. Nähere Informationen befinden sich im entsprechenden Abschnitt über die Befehlsstruktur.

Vor der Ausführung eines Single-Befehls muss der Anwender der IDENTControl unterschiedliche IN-Variablen konfigurieren. Durch den Parameter #HeadXDataFixcode wird festgelegt, ob ein Zugriff auf den Datenbereich (#HeadXDataFixcode = 0) oder auf den Fixcode (#HeadXDataFixcode = 1) durchgeführt wird. Ein Single-Befehl wird ausgeführt, wenn die Variable #HeadXSingleEnhanced nicht gesetzt ist. Der Befehl wird angestoßen, wenn die IN-Variablen #HeadXRead bzw. #HeadXWrite gesetzt sind. Durch das Setzen eines dieser beiden Parameter beginnt die Befehlszuweisung innerhalb des Funktionsbausteins. Anschließend wird das Ausgangsdatenfeld mit den Befehlsparametern an die IDENTControl übertragen und ausgeführt. Die IDENTControl sendet nach Erhalt der Befehlsparameter Statusrückmeldungen an die SPS zurück. Das Senden der Statusrückmeldungen wird solange durchgeführt, bis der Befehl vollständig abgearbeitet ist. Die Statusmeldung wird innerhalb des Funktionsbausteins dazu benutzt, um eine Fehlerauswertung durchzuführen. Die genaue Bedeutung der einzelnen Statuswerte ist dem Handbuch „IDENTControl IC-KP-B6-SUBD/-V15B“ (www.pepperl-fuchs.com) zu entnehmen.

Durch die Auswertung der Statusinformationen können kopfbezogene Statusbits generiert werden. Anhand dieser Statusbits kann eine Aussage über den Ausführungszustand des Befehls getroffen werden.

Das Statusbit #Head_X.Busy signalisiert, dass der Befehl noch von der IDENTControl ausgeführt wird. Wenn das Bit gesetzt ist, kann kein Befehl an diesem Kopf gestartet werden. Das Bit wird vom Funktionsbaustein zurückgesetzt, wenn die Abarbeitung des Befehls durch die IDENTControl beendet wurde.

Das Ende der Befehlsbearbeitung wird durch das Statusbit #Head_X.Done gekennzeichnet. Nach Beendigung der Befehlsausführung durch die IDENTControl wird dieses Bit selbstständig durch den Funktionsbaustein gesetzt. Eine erneute Befehlsausführung ist an einen Kopf erst dann möglich, wenn das Bit #Head_X.Busy nicht gesetzt und das Bit #Head_X.Done gesetzt ist. Befindet sich zum Zeitpunkt der Befehlsausführung kein Datenträger im Erfassungsbereich des Schreib-/Lesekopfes, so wird das Statusbit #Head_X.NoDataCarrier gesetzt. Daraufhin wird die Befehlsausführung automatisch von der IDENTControl wiederholt. Die maximale Anzahl der Wiederholungen wird durch die IN-Variable #RetrySingleCommand vom Anwender

vorgegeben. Die Anzahl der maximalen Wiederholungen gilt für alle Schreib-/Leseköpfe.

Wenn bei der Befehlsbearbeitung ein Timeout aufgetreten ist, so wird das Bit #Head_X.TimeoutOccured gesetzt. Ein Timeout tritt dann auf, wenn das Ende der Befehlsbearbeitung nicht innerhalb der durch die IN-Variable #Timeout vorgegebenen Zeitspanne erfolgte. Dies wird als Fehler betrachtet und das Bit #Head_X.Error gesetzt.

Das Bit #Head_X.Error signalisiert das Auftreten eines Fehlers in der Befehlsausführung der IDENTControl. Sobald das Bit #Head_X.Error gesetzt ist, ist der zugehörige Kopf für weitere Befehlsausführungen gesperrt. Die Sperre kann durch die IN-Variable #QuitErrorHeadX aufgehoben werden.

Die eingelesenen Nutzdatenblöcke befinden sich innerhalb des Funktionsbausteins in den Variablen #Head_X.InData.DW1…DW15. Nach Beendigung der Befehlsbearbeitung können die eingelesenen Daten weiterverarbeitet werden.

Die Datenblöcke, welche auf einen Datenträger geschrieben werden sollen, befinden sich in den Variablen #Head_X.OutData.DW1… DW15. Die Datenblöcke sind frei durch den Anwender festzulegen und müssen vor dem Start der Befehlsausführung zugewiesen werden. Die für die Ausführung eines Single-Befehls notwendigen Einstellungen sind der Befehlsliste im Anhang zu entnehmen.

2.1.6 Durchführung eines Enhanced-Befehls

Ein Enhanced-Befehl wird von der IDENTControl so lange ausgeführt, bis er durch einen Quit-Befehl abgebrochen wird. Bei einem Enhanced-Befehl wird der Lese- bzw. Schreibvorgang so lange durchgeführt, bis die Daten auf den Datenträger geschrieben bzw. vom Datenträger gelesen wurden. Im Gegensatz zum Single-Befehl bleibt der Enhanced-Befehl auch nach der Durchführung des Lese- bzw. Schreibvorgangs aktiv. Verlässt ein „alter“ Datenträger den Erfassungsbereich der Schreib-/Leseköpfe, so kann ein „neuer“ Datenträger innerhalb des Erfassungsbereiches ausgelesen oder beschrieben werden. Bei der Ausführung des Enhanced-Befehls ist zu beachten, dass sich immer nur ein Datenträger im Erfassungsbereich des Schreib-/Lesekopfes befindet.

Für die Ausführung von Enhanced-Befehlen müssen analog zum Single-Befehl verschiedene IN-Parameter festgelegt werden. Wie bei einem Single-Befehl wird durch die Variable #HeadXDataFixcode festgelegt, ob auf den Datenbereich oder Fixcode

zugegriffen werden soll. Mit Hilfe der Variablen #HeadXSingleEnhanced erfolgt die Ausführung eines Enhanced-Befehls. Die Variable muss vor Beginn der Befehlsausführung gesetzt werden. Durch die Parameter #HeadXRead und #HeadXWrite wird die Befehlsausführung analog zum Single-Befehl angestoßen.

Angestoßen wird die Befehlsausführung analog zum Single-Befehl durch die Parameter #HeadXRead und #HeadXWrite.

Der Befehl wird nun von der IDENTControl bearbeitet, was durch das Bit #Head_X.Busy angezeigt wird. Das Bit bleibt gesetzt, bis der Enhanced-Befehl durch einen Fehler oder einen Quit-Befehl abgebrochen wurde.

Das Bit #Head_X.Done signalisiert bei einem Enhanced-Befehl, dass ein Code-/Datenträger gelesen oder beschrieben wurde. Im Gegensatz zum Single-Befehl, wird hier nicht die Beendigung eines Befehls signalisiert. Entfernt sich der Code-/Datenträger wieder aus den Erfassungsbereich des Schreib-/Lesekopfes, wird das Bit #Head_X.Done automatisch zurückgesetzt, aber der Befehl ist noch aktiv.

Eine Ausführung eines neuen Befehls ist erst dann möglich, wenn das Bit #Head_X.Busy zurückgesetzt und das Bit #Head_X.Done gesetzt ist. Diese Bedingung wird nach der Ausführung eines Quit-Befehls erreicht.

Wie bei der Ausführung eines Single-Befehls befinden sich die Nutzdaten, die auf einen Datenträger geschrieben werden sollen, in den Variablen #Head_X.OutData.DW1… DW15. Die Ausgangsdaten müssen vor der Ausführung eines Enhanced-Befehls parametriert werden. Eine Änderung der Ausgangsdaten im laufenden Betrieb hat keine Auswirkung auf den laufenden Enhanced-Befehl.

Die Eingangsdaten werden in den Variablen #Head_X.InData.DW1… DW15 innerhalb des Instanz-DB abgelegt. Die notwendigen Einstellungen für die Ausführung eines Enhanced-Befehls sind der Befehlsliste im Anhang zu entnehmen.

2.1.7 Durchführung eines Special-Command

Mit Hilfe eines Special-Command kann der Anwender einen Befehl selbstständig parametrieren. Der Special-Command kann in erster Linie für die Durchführung von Befehlen genutzt werden, die nicht in der Befehlsliste des Funktionsbausteins enthalten sind. Die Ausführung von Standardbefehlen z.B. Single-Read/-Write ist über den Special-Command allerdings ebenfalls möglich.

Vor der Durchführung eines Special-Commands müssen die Befehlsparameter in ein eigens dafür vorgesehenes Datenfeld (#Head_X.SpecialCommand) innerhalb des

Instanz-Datenbausteins übergeben werden. Der Befehlscode des auszuführenden Befehls muss dabei der Variable #Head_X.SpecialCommand.Code zugewiesen werden. Eine Kanalzuweisung ist für die Ausführung eines Special-Command am entsprechenden Schreib-/Lesekopf nicht notwendig. Die entsprechende Kanalzuweisung wird innerhalb des Funktionsbausteins automatisch durchgeführt. Bei der Ausführung eines Read-/Write-Befehls mit Hilfe eines Special-Commands muss in der Variablen #Head_X.SpecialCommand.Channel die Anzahl der zu übertragenden Nutzdatenwörter angegeben werden. Dazu werden die oberen vier Bit der Variablen genutzt.

Für die Parametrierung eines Special-Command stehen weitere Variablen zur Verfügung. Durch die Variablen #Head_X.SpecialCommand.Parameter1… 6 können weitere Befehlsparameter zugewiesen werden.

Nähere Informationen über die Befehlsparametrierung sind dem Handbuch „IDENTControl IC-KP-B6-SUBD/-V15B“ in der Befehlsübersicht zu entnehmen.

Die Ausführung eines Special-Command wird durch das Setzen der IN-Variable #HeadXSpecialCommand gestartet. Daraufhin wird das Datenfeld #Head_X.SpecialCommand dem Ausgangsdatenfeld #Head_X.OutData des zugehörigen Kopfes übergeben und an die IDENTControl gesendet. Der übermittelte Befehl wird anschließend von der IDENTControl ausgeführt.

Wenn durch den Special-Command ein Enhanced-Befehl gestartet wurde, muss er wieder über einen Quit-Befehl am zugehörigen Kopf abgebrochen werden.

Mit Hilfe des Special-Command können auch Befehle an die IDENTControl gesendet werden, die nicht von den Schreib-/Leseköpfen, sondern nur von der IDENTControl bearbeitet werden. Diese Befehle werden als IDENTControl-Befehle bezeichnet. Ein Beispiel für einen IDENTControl-Befehl ist der Befehl Set-Multiplexed-Mode. Wie bei einem Special-Command erfolgt die Parametrierung über das Datenfeld #Head_1.SpecialCommand. Da die Befehle nicht von den Schreib-/Leseköpfen ausgeführt werden, ist keine Kanalzuweisung erforderlich. Die Ausführung des Befehls wird durch das Setzen der IN-Variablen #Head1SpecialCommand und #IC_Command_on_Head1 gestartet. Daraufhin wird das Datenfeld #Head_1.SpecialCommand an das Ausgangsdatenfeld #Head_1.OutData übergeben und an die IDENTControl übertragen und ausgeführt.

2.1.8 Durchführung einer Fehlerauswertung

Bei der Ausführung eines Befehls kann ein Fehler auftreten. Der Funktionsbaustein bietet die Möglichkeit eine Fehlerauswertung durchzuführen. Wenn bei der Ausführung eines Befehls ein Fehler auftritt, wird das Bit #Head_X.Error gesetzt. Der Kopf ist daraufhin für eine weitere Befehlsausführung gesperrt. Für die Fehlerauswertung stehen verschiedene Parameter zur Verfügung.

In der nachfolgenden Tabelle sind die verschiedenen Parameter zur Fehlerauswertung aufgeführt.

PARAMETER	BEDEUTUNG
#Head_X.Error	Fehler in der Befehlsausführung
#Head_X.InvalidResponse	Ungültige Antwort der IDENTControl
#Head_X.TimeoutOccured	Timeout abgelaufen
#Head_X.Error_SFC_14	Fehler bei der Ausführung SFC 14
#Head_X.Error_SFC_15	Fehler bei der Ausführung SFC 15
#Head_X.RetVal_SFC14	Fehlercode SFC 14
#Head_X.RetVal_SFC15	Fehlercode SFC 15
#Memory.Error_SFC14	Fehler bei der Ausführung SFC 14
#Memory.RetVal_SFC14	Fehlercode SFC 14
#RetVal_SFC20	Fehlercode SFC 20

Tab. 6: Fehlerauswertung

Die Fehlerverriegelung eines Schreib-/Lesekopfes kann über den Quit-Error-Befehl aufgehoben werden. Der Quit-Error Befehl wird durch das Setzen der IN-Variable #QuitErrorHeadX gestartet. Durch diesen Befehl werden alle Bits zurückgesetzt, die eine Befehlsausführung anderer Befehle unterbinden. Dadurch können Befehle am entsprechenden Schreib-/Lesekopf ausgeführt werden, sofern kein weiterer Fehler eine Verriegelung verursacht.

2.2 Verwendete Bausteine und ihre Funktionalität

Die Anbindung einer IDENTControl an eine SPS wird über einen Funktionsbaustein realisiert. Ein Funktionsbaustein ist frei programmierbar und kann somit optimal auf die Anwendung zugeschnitten werden. Die für die Bearbeitung eines Funktionsbausteins anfallenden Daten werden in einem Instanz-Datenbaustein abgespeichert. Ebenso sind noch weitere verschiedene Systemfunktionen unterschiedlicher Funktionalität für die Realisierung der Kommunikation zwischen IDENTControl und SPS erforderlich. Nachfolgende Tabelle gibt einen Überblick über die verwendeten Bausteine und ihrer Funktion.

Baustein	Typ	Funktion
OB1	Organisationsbaustein	Wird vom Betriebssystem zyklisch durchlaufen

FB 32/64 „IDENTControl“	Funktionsbaustein	Steuerung des Kommunikationsablaufs mit der IDENTControl
DB 32/64 „InstDB“	Instanzen-Datenbaustein	Speicherung der anfallenden Lokaldaten
SFC 14 „DPRD_DAT"	Systemfunktion	Einlesen von Daten eines DP-Slaves
SFC 15 „DPWR_DAT"	Systemfunktion	Senden von Daten an einen DP-Slave
SFC 20 „BLKMOV“	Systemfunktion	Umkopieren eines Speicherbereiches
SFB 5 „TOF“	Systemfunktionsbaustein	Erzeugung einer Ausschaltverzögerung

Tab. 7: Verwendete Bausteine und ihre Funktion

Der Zusammenhang zwischen den einzelnen Bausteinen wird in der nachfolgenden Grafik dargestellt.

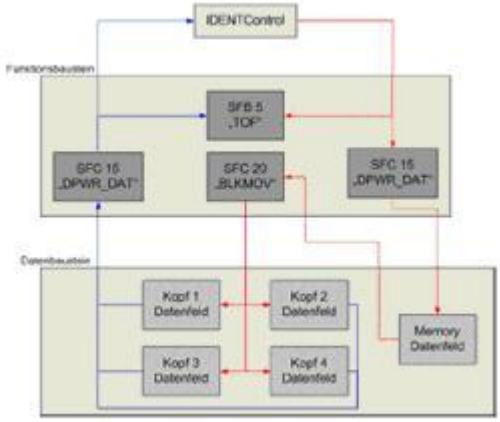

Abb 8: Zusammenhang der Bausteine

In dieser Grafik sind neben den verwendeten Bausteinen auch die Datenströme veranschaulicht, welche zwischen der SPS und der IDENTControl ausgetauscht werden. Die jedem Kopf zugeordneten Datenfelder sind in ein Eingangs- und ein Ausgangsdatenfeld untergliedert. Das Senden der Daten von der SPS zur IDENTControl wird durch die Systemfunktion SFC 15 durchgeführt. Die Funktion sendet Daten über eine projektierte Profibus-DP Verbindung an einen DP-Slave. Beim Senden der Daten wird gleichzeitig die Ausschaltverzögerung (SFB 5) aktiviert, um somit die Antwortzeit der IDENTControl zu überwachen. Der Empfang der Daten wird durch die Systemfunktion SFC 14 realisiert. Die Ausschaltverzögerung wird deaktiviert, sobald Daten von der IDENTControl eingelesen wurden. Die eingelesenen Daten werden in einem Memorydatenfeld innerhalb des Instanz-DB abgespeichert. Die Daten innerhalb des Memorydatenfeldes werden analysiert und danach durch die Systemfunktion SFC 20 in das Eingangsdatenfeld des zugehörigen Kopfes kopiert.

2.3 Organisationsbaustein OB 1

Der Organisationsbaustein OB 1 wird von dem Betriebssystem der CPU zyklisch durchlaufen. Der OB 1 bildet die Schnittstelle zwischen dem Betriebssystem der CPU und dem Anwenderprogramm.

Zu Beginn des OB 1 wird zunächst die Datenträgerkennung in den Instanz-Datenbaustein geladen. Die Zuweisung der Datenträgerkennung ist für die Ausführung der Initialisierung erforderlich und muss vor der Ausführung der Initialisierung durchgeführt werden. Bei der Zuweisung der Datenträgerkennung ist darauf zu achten, dass die Angabe der Kennung in hexadezimaler Form erfolgt. Die Datenträgerkennung wird der Variablen #Head_X.TagType übergeben. Die Zuweisung der Datenträgerkennung muss für jeden angeschlossenen Schreib-/Lesekopf durchgeführt werden.

Anschließend erfolgt die Zuweisung der Anzahl der zu übertragenden Nutzdatenwörter. Die Anzahl der maximal übertragbaren Nutzdatenwörter ist abhängig von der in der Hardware-Konfiguration getroffenen Porteinstellung. Der Anwender kann aber die Anzahl in den vorgegebenen Grenzen frei bestimmen. Die Zuweisung wird über den Parameter #Head_X.WordNum durchgeführt. Dabei ist zu beachten, dass die Zuweisung immer vor dem Start eines Befehles durchgeführt und für die Zeit der Befehlsausführung nicht verändert wird.

Im nächsten Schritt wird der Funktionsbaustein „IDENTControl“ und der zugehörige Instanz-Datenbaustein „InstDB“ aufgerufen. Die Bausteine werden über den Bausteinaufruf „Call“ gestartet.

Beispiel: Call „IDENTControl“ , „InstDB“

Der Funktionsbaustein ist multiinstanzfähig. Darunter ist zu verstehen, dass einem Funktionsbaustein mehrere Instanz-Datenbausteine zugewiesen werden können. Das ermöglicht die Anbindung mehrerer IDENTControl in eine Steuerung. Dazu muss der Baustein mehrmals aufgerufen werden.

Beispiel: Call „IDENTControl“ , „InstDB1“
Call „IDENTControl“ , „InstDB2“

2.4 Funktionsbaustein „IDENTControl“

Der Funktionsbaustein „IDENTControl“ hat die Aufgabe, das Identifikationssystem IDENTControl in die Steuerung einzubinden. Ein Funktionsbaustein ist durch den Anwender frei programmierbar. Dadurch kann die Funktionalität der IDENTControl auf die Anwendung angepasst werden.

Der Funktionsbaustein ist in verschiedene Netzwerke untergliedert. Nachfolgende Tabelle stellt die verschiedenen Netzwerke sowie deren Aufgabe innerhalb des FBs dar.

NETZWERK	AUFGABE
1	Start-Up-Sequence
2	Initialisierung Kopf 1
3	Initialisierung Kopf 2
4	Initialisierung Kopf 3
5	Initialisierung Kopf 4
6	Timeout-Überwachung
7	Umsetzung von IN- auf STAT-Variablen
8	Einlesen der Rückantwort
9	Befehlszuweisung Kopf 1
10	Befehlszuweisung Kopf 2
11	Befehlszuweisung Kopf 3
12	Befehlszuweisung Kopf 4
13	Befehlsausführung Kopf 1
14	Befehlsausführung Kopf 2
15	Befehlsausführung Kopf 3
16	Befehlsausführung Kopf 4
17	Restart-Routine
18	Quittieren
19	Auswertung SFC 15
20	Auswertung SFC 14
21	Analyse der Eingangsdatenfelder

Tab. 8: Netzwerke des Funktionsbausteins

2.4.1 Ablauf der Start-UP-Sequence

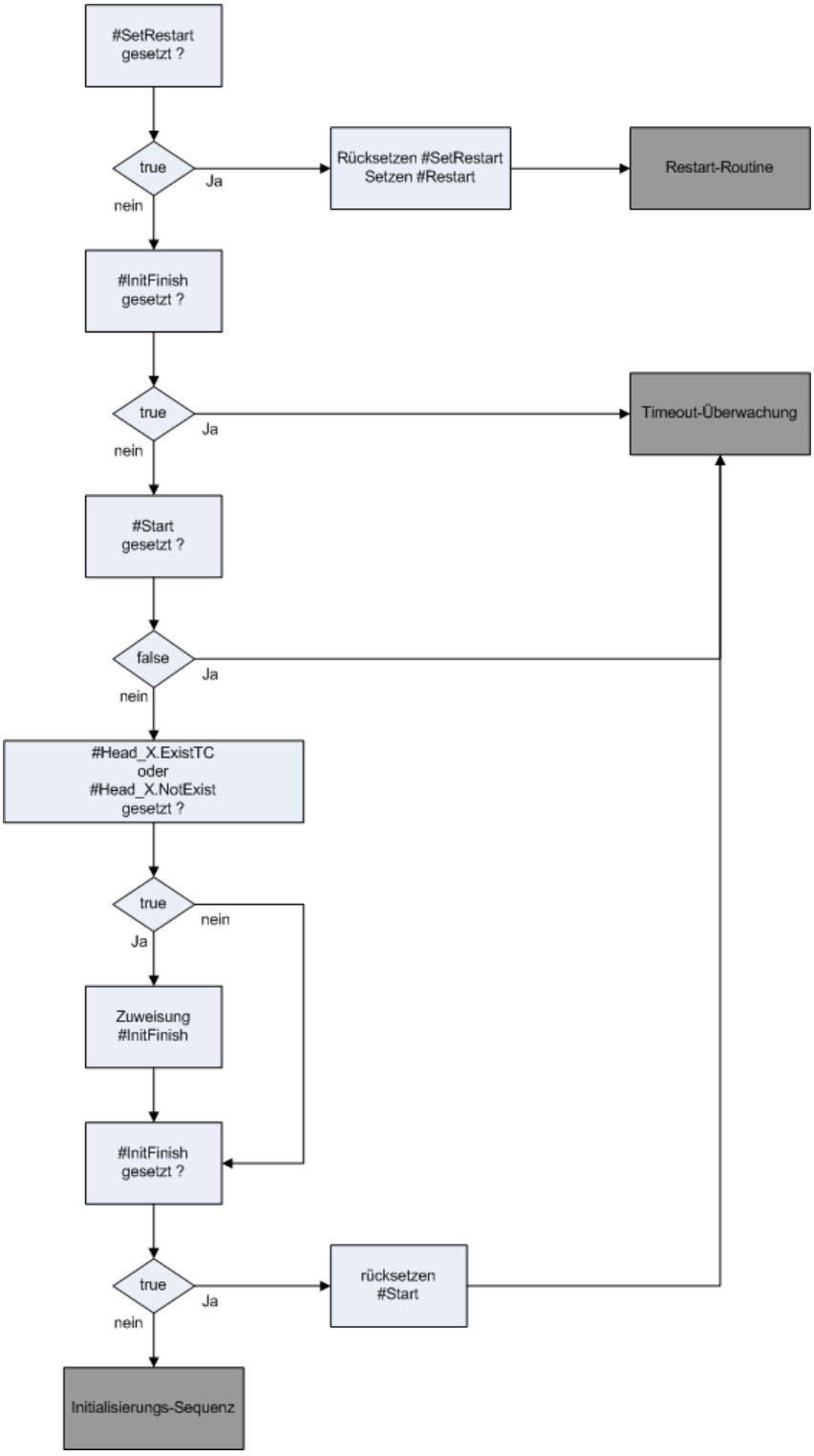

Abb. 9: Ablaufplan Start-Up-Sequence

Die Start-UP-Sequence im Netzwerk 1 hat die Aufgabe festzustellen, ob die Schreib-/Leseköpfe der IDENTControl initialisiert worden sind. Mit Hilfe der Initialisierung wird überprüft, an welchen Kanälen die Schreib-/Leseköpfe an die IDENTControl angeschlossen sind.

Zu Beginn wird überprüft, ob das Bit #SetRestart gesetzt ist. Dadurch ist es möglich, einen Neustart des Programms durchzuführen. Ist die Bedingung für einen Neustart erfüllt, erfolgt ein Sprung zur Sprungmarke sres in das Netzwerk 17.
Ist ein Neustart des Programms nicht vorgesehen, wird das Bit #InitFinish überprüft. Ist es gesetzt, so ist die Initialisierung aller Köpfe bereits abgeschlossen und es erfolgt ein Sprung zur Sprungmarke end in das Netzwerk 6 zur Timeoutüberwachung. Ebenso erfolgt ein Sprung zur Timeoutüberwachung, wenn das Bit #Start nicht gesetzt ist. Mit Hilfe des #Start Bits wird signalisiert, dass ein Neustart des Programms durchgeführt worden ist.
Im nachfolgenden Abschnitt erfolgt die Überprüfung, inwieweit die Initialisierung für alle Kanäle erfolgreich abgeschlossen ist. Dazu wird geprüft, ob das Bit #Head_X.Exist oder #Head_X.NotExist gesetzt ist. Sobald eines der beiden Bits gesetzt ist, wurde die Initialisierung für den Kanal erfolgreich durchgeführt. Dabei ist zu unterscheiden, ob ein Schreib-/Lesekopf an den entsprechenden IDENT-Kanal angeschlossen (#Head_X.Exist) oder nicht angeschlossen (#Head_X.NotExist) ist. Die Überprüfung erfolgt für alle Köpfe. Ist die Überprüfung für alle Köpfe erfolgreich abgeschlossen, wird das Bit #InitFinish gesetzt. Dieses Bit signalisiert die erfolgreiche Beendigung der Initialisierung für alle Köpfe. Abschließend erfolgt das Rücksetzen des #Start Bits. Ist das Bit #InitFinish nicht gesetzt, wird eine Initialisierung der Köpfe durchgeführt.

2.4.2 Ablauf der Initialisierung

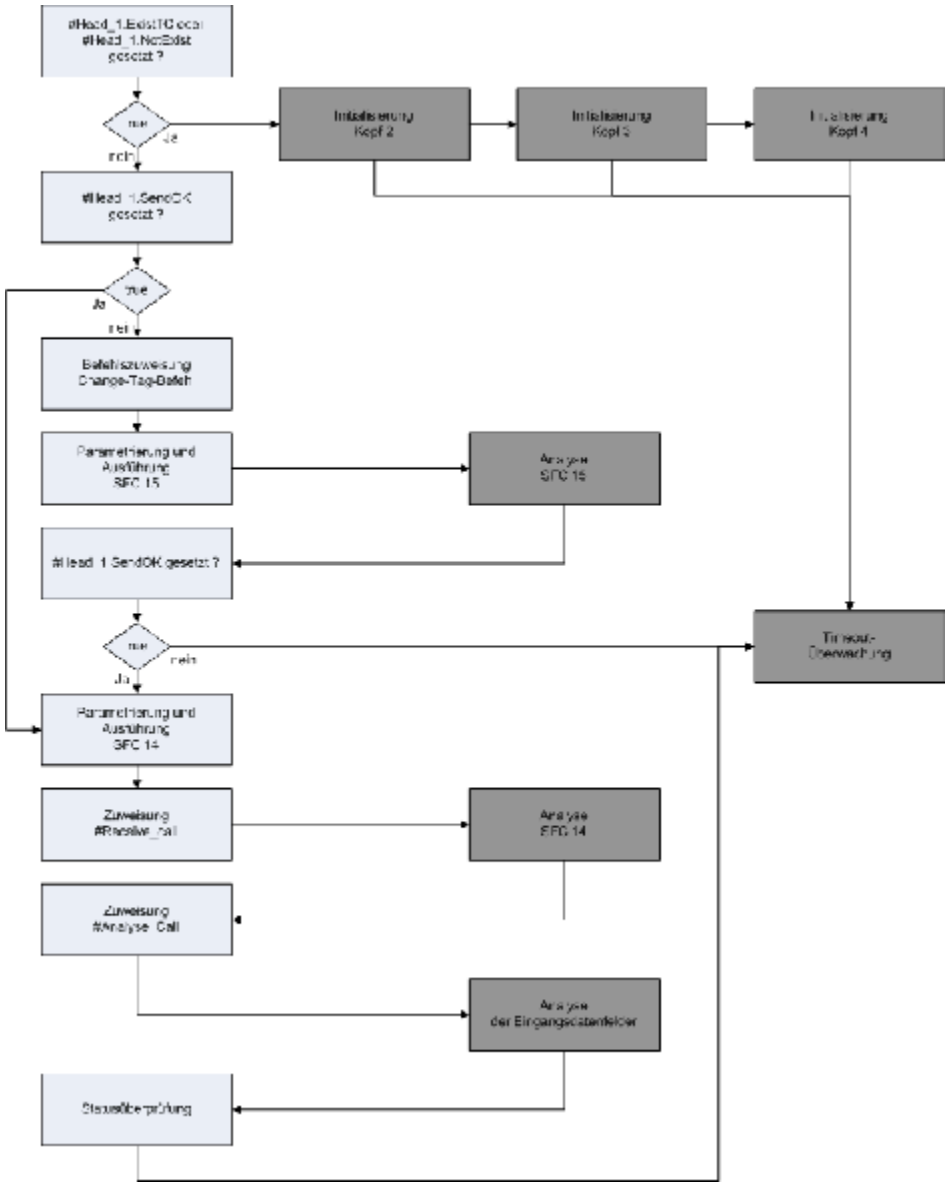

Abb. 10: Ablaufplan der Initialisierung von Kopf 1

Der Programmabschnitt Initialisierung ist erforderlich um festzustellen, ob ein Schreib-/Lesekopf an einen IDENT-Kanal angeschlossen ist. Bei der Initialisierung wird ein Change-Tag Befehl an den entsprechenden IDENT-Kanal der Auswerteeinheit geschickt. Mit Hilfe dieses Befehls wird dem Schreib-/Lesekopf mitgeteilt, welcher Code-/Datenträger beschrieben bzw. ausgelesen werden soll. Nachfolgend ist die Initialisierung des IDENT-Kanal 1 beschrieben. Für die anderen IDENT-Kanäle erfolgt die Initialisierung analog zu diesem Teil.

Zu Beginn des Programmteils wird überprüft, ob der IDENT-Kanal bereits initialisiert wurde. Dazu werden die Bits #Head_1.Exist und #Head_1.NotExist überprüft. Ist eines dieser beiden Statusbits gesetzt, so ist die Initialisierung für diesen Kanal bereits

erfolgt und es wird zur Sprungmarke ini2 gesprungen. An dieser Sprungmarke beginnt die Initialisierung für den IDENT-Kanal 2.

Ist die Initialisierung für diesen Kopf noch nicht erfolgreich abgeschlossen, wird zunächst überprüft, ob das Bit #Head_1.SendOK gesetzt ist. Mit Hilfe dieses Bits wird an dieser Stelle signalisiert, dass der Change-Tag Befehl bereits an die IDENT-Control geschickt wurde. Damit der Befehl nicht erneut gesendet wird, erfolgt ein Sprung zur Sprungmarke rec1 im selben Netzwerk. Durch diesen Sprung wird das Laden der Befehlsparameter für den Change-Tag-Befehl in das Ausgangsdatenfeld sowie das erneute Senden dieses Befehls an die IDENTControl unterbunden.

Im Anschluss an die Abfrage der Statusbits werden die Befehlsparameter für den Change-Tag-Befehl in das Ausgangsdatenfeld der IDENTControl übergeben.

Nach der Übergabe der Befehlsparameter werden verschiedene Statusbits gesetzt bzw. zurückgesetzt. Durch das Setzen von #Head_1.Busy wird gekennzeichnet, dass am Kanal 1 gerade ein Befehl bearbeitet wird. Mit #Head_1.TimeoutActive wird die Timeoutüberwachung zur Ausführung des in das Ausgangsdatenfeld geladenen Befehls vorbereitet.

Nachdem die Befehlsparameter in das Ausgangsdatenfeld übertragen und die Statusbits gesetzt wurden, erfolgt das Senden des Befehls an die IDENTControl. Das Senden der Daten an die IDENTControl über Profibus erfolgt durch die Systemfunktion SFC 15. Die für die Ausführung der Funktion erforderlichen Parameter werden mit Hilfe eines ANY-Parameters der Funktion zugeführt. Anschließend erfolgt der Aufruf der Funktion SFC 15. Nach erfolgreicher Ausführung der Funktion erfolgt ein Sprung zur Sprungmarke aus1 in das Netzwerk 19. Dort erfolgt eine Fehlerauswertung für die Funktion.

Nachdem die Fehlerauswertung durchgeführt wurde und kein Fehler bei der Ausführung der Funktion aufgetreten ist, erfolgt der Rücksprung zur Sprungmarke F151 in das Netzwerk 2.

An der Sprungmarke F501 wird überprüft, ob das Bit #Head_1.SendOK gesetzt ist. Dieses Bit ist gesetzt, wenn die Funktion SFC 15 erfolgreich und ohne Fehler ausgeführt wurde. In diesem Fall erfolgt die Weiterverarbeitung des Programms an der Sprungmarke rec1. Ist das Bit #Head_1.SendOK nicht gesetzt, erfolgt ein Sprung zur Timeoutüberwachung in das Netzwerk 6.

Nachdem das Senden der Daten an die IDENTControl erfolgreich durchgeführt worden ist, erfolgt im Anschluss das Empfangen der Rückantwort von der IDENTControl.

Das Empfangen von Daten über Profibus wird durch die SPS über die Systemfunktion SFC 14 durchgeführt.

An der Sprungmarke rec1 werden die für die Ausführung der Funktion erforderlichen Parameter mit Hilfe eines ANY-Parameters der Funktion zugeführt. Anschließend erfolgt der Aufruf der Funktion SFC 14. Abschließend wird der Wert 0 der Variablen #Receive_call zugewiesen. Nach der Zuweisung erfolgt ein Sprung zur Sprungmarke aus5 in das Netzwerk 20. Dort findet eine Fehlerauswertung für die Funktion statt. Nachdem die Fehlerauswertung durchgeführt wurde und kein Fehler bei der Ausführung aufgetreten ist, erfolgt der Rücksprung zur Sprungmarke F141 in das Netzwerk 2.

An der Sprungmarke F141 wird der Variablen #Analyse_Call der Wert 0 zugewiesen. Durch diese Variable erfolgt mit Hilfe einer Sprungauswahlliste der Rücksprung an die Sprungmarke ana1 in diesem Netzwerk. Danach erfolgt ein Sprung zur Sprungmarke lyse in das Netzwerk 21. Dort werden die von der IDENTControl empfangenen Daten analysiert. Nach Abschluss der Analyse findet der Rücksprung über eine Sprungauswahlliste in das Netzwerk 2 zur Sprungmarke ana1 statt.

In diesem Programmabschnitt erfolgt die Auswertung der Statusrückmeldung. Die Antwort der IDENTControl auf einen Befehl enthält eine Statusmeldung. Durch deren Auswertung kann man Rückschlüsse auf die Ausführung des Befehls ziehen. Die Statusrückmeldung ist ein ein Byte großer Code, welcher in der Variablen #Head_1.InData.Status enthalten ist. Hat der Status den Wert (06)h, so ist kein Schreib-/Lesekopf an den IDENT-Kanal 1 angeschlossen und somit wird das Bit #Head_1.NotExist gesetzt. Auf Grund dessen, dass kein Schreib-Lesekopf angeschlossen ist, werden die Statusbits #Head_1.ExistTC und #Head_1.Error zurückgesetzt. Anschließend wird überprüft, ob das Statusbit #Head_1.Done gesetzt ist. Durch dieses Bit wird signalisiert, dass neue Daten zur Analyse am Kopf 1 bereitstehen. Ebenso muss die Überprüfung erfolgen, dass das Bit #Head_1.NotExist nicht gesetzt ist. Sind beide Bedingungen erfüllt, ist ein Schreib-Lesekopf an den IDENT-Kanal 1 angeschlossen und es wird das Statusbit #Head_1.ExistTC gesetzt. Abschließend erfolgt ein absoluter Sprung zur Sprungmarke end in das Netzwerk 6 zur Timeoutüberwachung. Nach Bearbeitung dieses Programmabschnitts ist die Initialisierung für den Kopf 1 abgeschlossen und es erfolgt im nächsten Zyklus die Initialisierung für Kopf 2, sofern sie noch nicht durchgeführt wurde.

2.4.3 Ablauf der Timeoutüberwachung

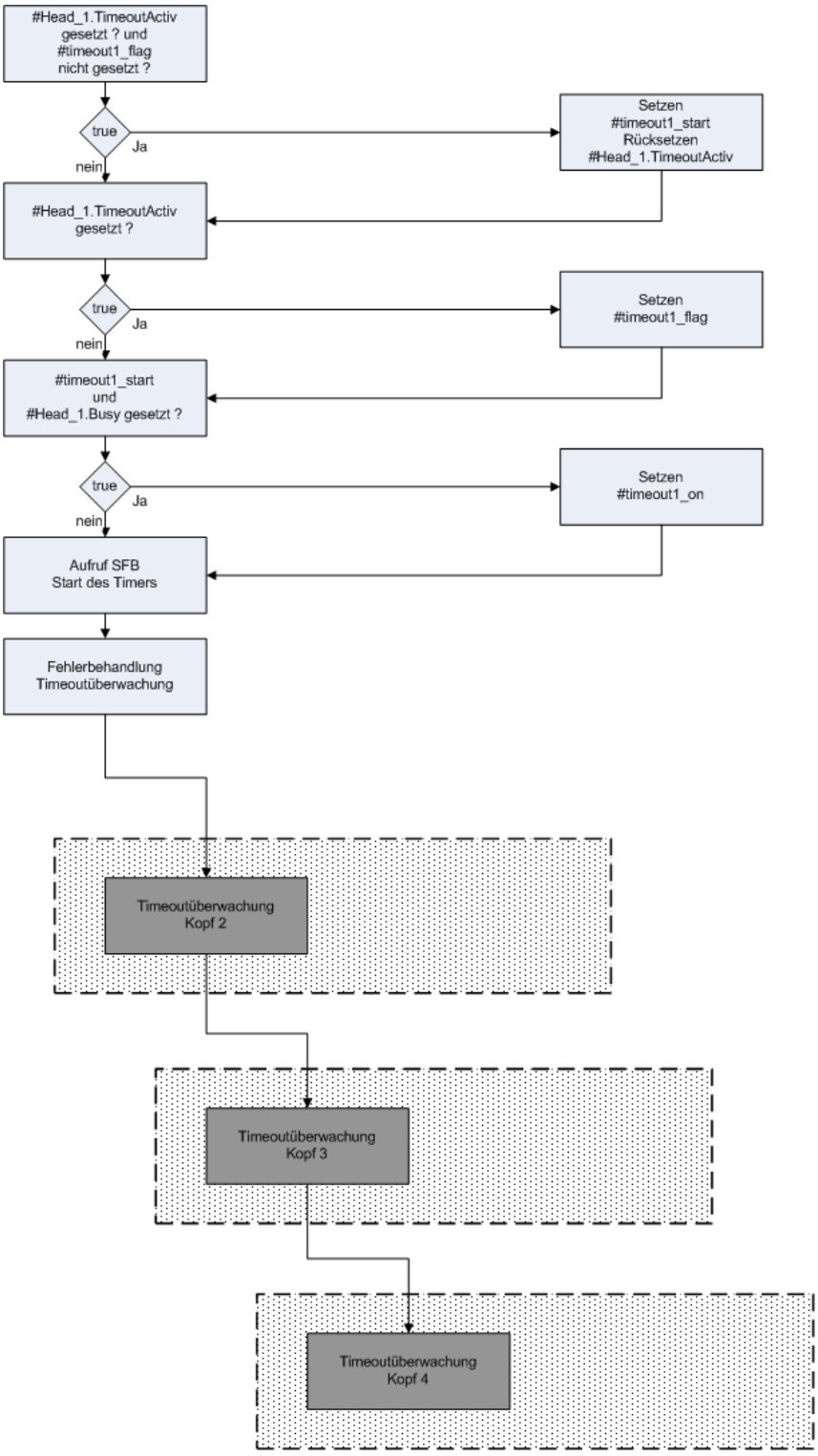

Abb. 11: Ablaufplan Timeoutüberwachung

Die Timeoutüberwachung hat die Aufgabe, die maximale Antwortzeit der IDENT-Control auf einen Befehl zu überwachen und bei einer zeitlichen Überschreitung eine

Fehlermeldung auszugeben. Nachfolgend wird die Timeoutüberwachung für Kopf 1 erläutert. Sie gilt analog auch für die anderen Köpfe.

Zu Beginn wird überprüft, ob das Bit #Head_1.TimeoutActiv gesetzt und das Bit #timeout1_flag nicht gesetzt ist. Das Verknüpfungsergebnis dieser Abfrage wird der Variablen #timeout1_start zugewiesen. Ist die Bedingung erfüllt, so wird die Variable #Head_1.TimeoutActiv zurückgesetzt. Andernfalls wird überprüft, ob nur die Variable #Head_1.TimeoutActiv gesetzt ist. Das Verknüpfungsergebnis wird der Variablen #timeout1_flag zugewiesen. Anschließend wird überprüft ob die Variablen #timeout1_start und #Head_1.Busy gesetzt sind. Das Verknüpfungsergebnis wird der Variablen #timeout1_on zugewiesen.

Im Anschluss erfolgt der Aufruf des SFB 5. Dieser erzeugt eine Ausschaltverzögerung am Ausgang Q. Die Variable bewirkt eine steigende Flanke am Eingang IN, was zu einer steigende Flanke am Ausgang Q bei der Variablen #timeout1_running führt. Liegt eine fallende Flanke am Eingang Q an, so entsteht eine fallende Flanke am Ausgang Q erst nach Ablauf der Zeit PT #Timeout.

Im Anschluss wird überprüft, ob das Bit #Head_1.Busy gesetzt sowie die Bits #Head_1.ReceivedOK, #timeout1_running und #timeout1_start nicht gesetzt sind. Ist diese Bedingung erfüllt, ist die Überwachungszeit überschritten und es erfolgt Fehlermeldung. Dabei werden die Bits #Head_1.TimeoutOccured, #Head_1.Error sowie #Head_1.Done gesetzt. Des Weiteren werden die Bits #Head_1.SglCommandActiv, #TransfToHead1, #Head_1.Busy sowie #Head_1.TimeoutActiv zurückgesetzt.

Dieses Netzwerk bewirkt eine Ausschaltverzögerung. Die Ausschaltbedingung wird realisiert, wenn #Head_1.TimeoutActiv und #timeout1_flag nicht gesetzt sind. Des Weiteren muss das Bit #Head_1.Busy gesetzt sein. Beide Head_1 Bits werden beim Aufruf der SFC 15 gesetzt. Durch die Variable #timeout1_flag wird signalisiert, dass ein Timeout noch aktiv ist und nicht verlängert werden kann.

Die Timeoutüberwachung für die anderen Köpfe erfolgt analog. An die Timeoutüberwachung schließt sich die Variabelenumsetzung von IN- auf STAT-Variablen an.

2.4.4 Ablauf der Variablenumsetzung von IN- auf STAT-Variablen

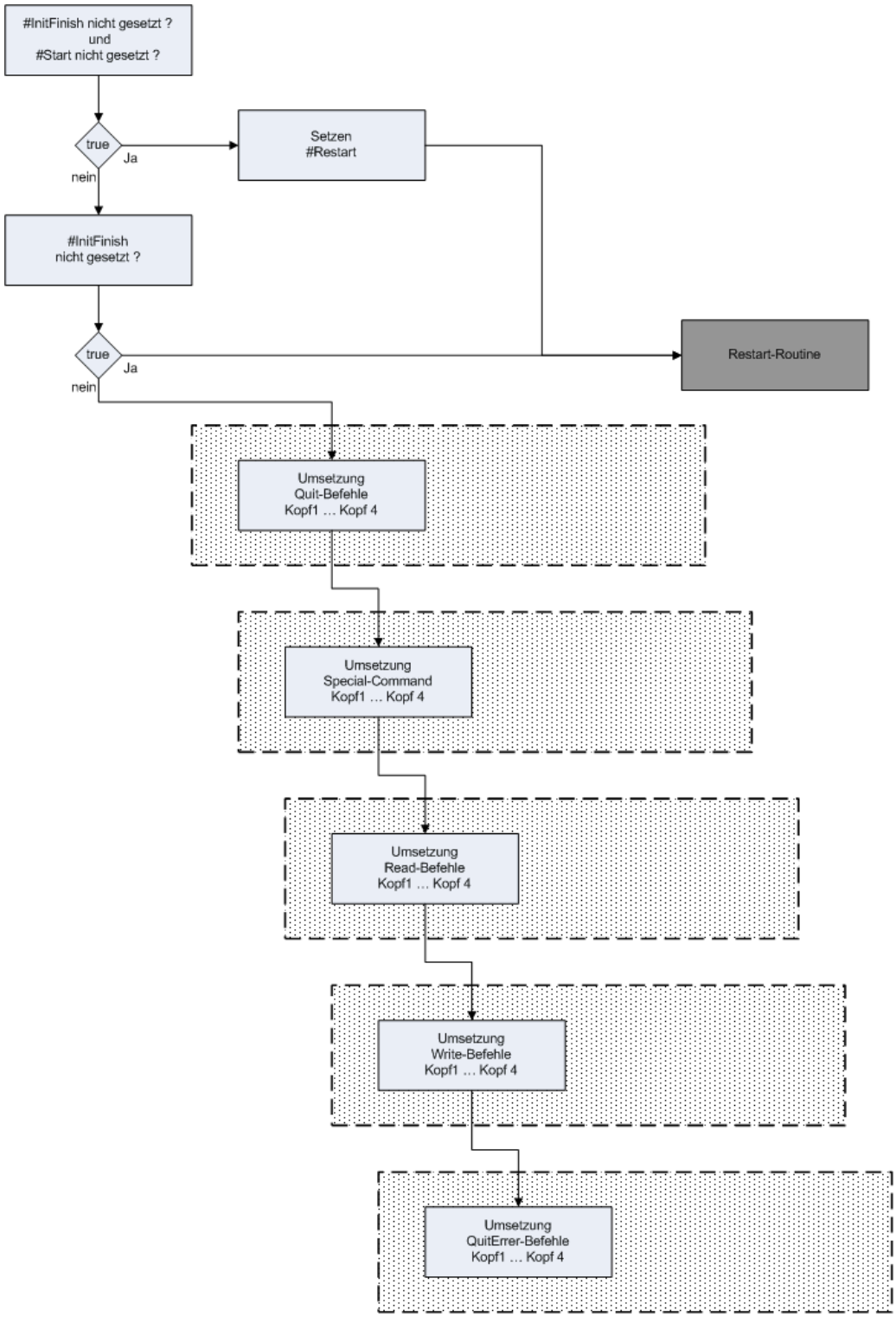

Abb. 12: Ablaufplan Variablenumsetzung

Dieser Abschnitt hat die Aufgabe, die Eingangsvariablen auf statische Variablen umzusetzen. Die Umsetzung ist erforderlich, weil sich während der Bearbeitung des Funktionsbausteins der Zustand der Eingangsvariablen ändern kann. Dadurch kann

ein Befehl angestoßen werden, obwohl noch ein anderer Befehl in Bearbeitung ist. Die IN-Variablen haben den Nachteil, dass sie durch den Funktionsbaustein nicht verändert werden können. Dadurch würde der Befehl, welcher durch eine IN-Variable angestoßen wird, solange ausgeführt, bis die IN-Variable durch einen Programmteil außerhalb des Funktionsbausteins zurückgesetzt wird. Eine Flankensteuerung verschafft an dieser Stelle eine Abhilfe.

Bei einer positiven Flanke im Signalverlauf einer IN-Variablen erfolgt die Umsetzung auf eine entsprechende STAT-Variable. Die STAT-Variable ist eine Lokalvariable innerhalb des Funktionsbausteins. Die STAT-Variablen haben den Vorteil, dass sie durch den Funktionsbaustein verändert werden können. Nach erfolgter Befehlsausführung wird die STAT-Variable zurückgesetzt und es kann wieder ein neuer Befehl angestoßen werden.

Zu Beginn des Netzwerks wird überprüft, dass die Variablen #InitFinish und #Start nicht gesetzt sind. Die Bedingung ist erfüllt, wenn die Initialisierung der Köpfe noch nicht abgeschlossen ist und kein Restart erfolgte. Somit wird das Bit #Restart gesetzt und es erfolgt ein Sprung zur Sprungmarke end1 in das Netzwerk 16. Dort wird die Restart-Routine durchgeführt.

Andernfalls erfolgt die alleinige Abfrage des Bits #InitFinish. Der Anwender kann durch das Rücksetzen dieses Bits die automatische Neuinitialisierung des Funktionsbausteins einleiten. Wurde die Initialisierung noch nicht durchgeführt, erfolgt ein Sprung zur Restart-Routinen, an welche sich die Initialisierung bei einem neuerlichen Programmdurchlauf anschließt.

Wurde die Initialisierung zuvor bereits erfolgreich abgeschlossen, erfolgt nun die eigentliche Befehlsumsetzung.

Anschließend werden die extern gegebenen Quit-Befehle auf interne Signale umgewandelt. Es wird der vorige Signalzustand in der Variablen #SaveQuitHead1 gespeichert. Diese Variable wird in jedem Zyklus mit der Verknüpfungsvariablen #Head1Quit verglichen. Geht die Variable #Head1Quit auf „1“ und war der Vorzustand „0“ (in #SaveQuitHead1 gespeichert), so wird ein positiver Flankenwechsel registriert und das Bit #QuitHead1 gesetzt.

Die Umsetzung der anderen Befehle erfolgt nach dem gleichen Muster.

2.4.5 Ablauf Programmteil Einlesen der Daten

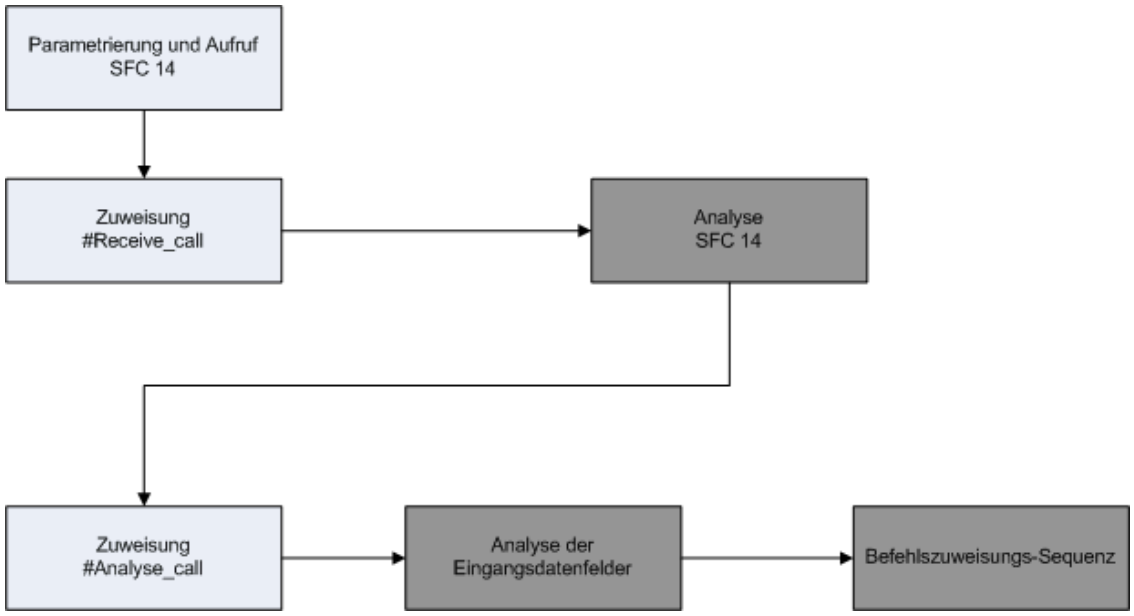

Abb. 13: Ablaufplan Einlesen von Daten

In diesem Netzwerk erfolgt das Einlesen der Daten durch die Funktion SFC 14. Zu Beginn werden die zur Ausführung der Funktion SFC 14 erforderlichen Parameter über einen ANY-Parametertyp zugewiesen. Anschließend wird die Funktion aufgerufen und Daten werden von der projektierten Slave-Adresse eingelesen. Danach wird der Variablen #Receive_call der Wert 4 zugewiesen und es erfolgt ein Sprung zur Sprungmarke aus5 in das Netzwerk 20. Dort werden die eingelesenen Daten analysiert und in die zugehörigen Eingangsdatenfelder umkopiert. Mit Hilfe der Variablen #Receive_call erfolgt anschließend der Rücksprung in das Netzwerk 8 an die Sprungmarke F145. Hier wird der Variablen #Analyse_call der Wert 4 zugewiesen und zur Sprungmarke lyse in das Netzwerk 21 gesprungen. Dort werden die Daten der Eingangsdatenfelder der Köpfe analysiert. Nach der Analyse der Eingangsdatenfelder erfolgt ein Sprung zur Sprungmarke ana5 vor das Netzwerk 9. Dort beginnt die Befehlszuweisung der Köpfe.

2.4.6 Ablauf der Befehlszuweisungssequenz von Kopf 1

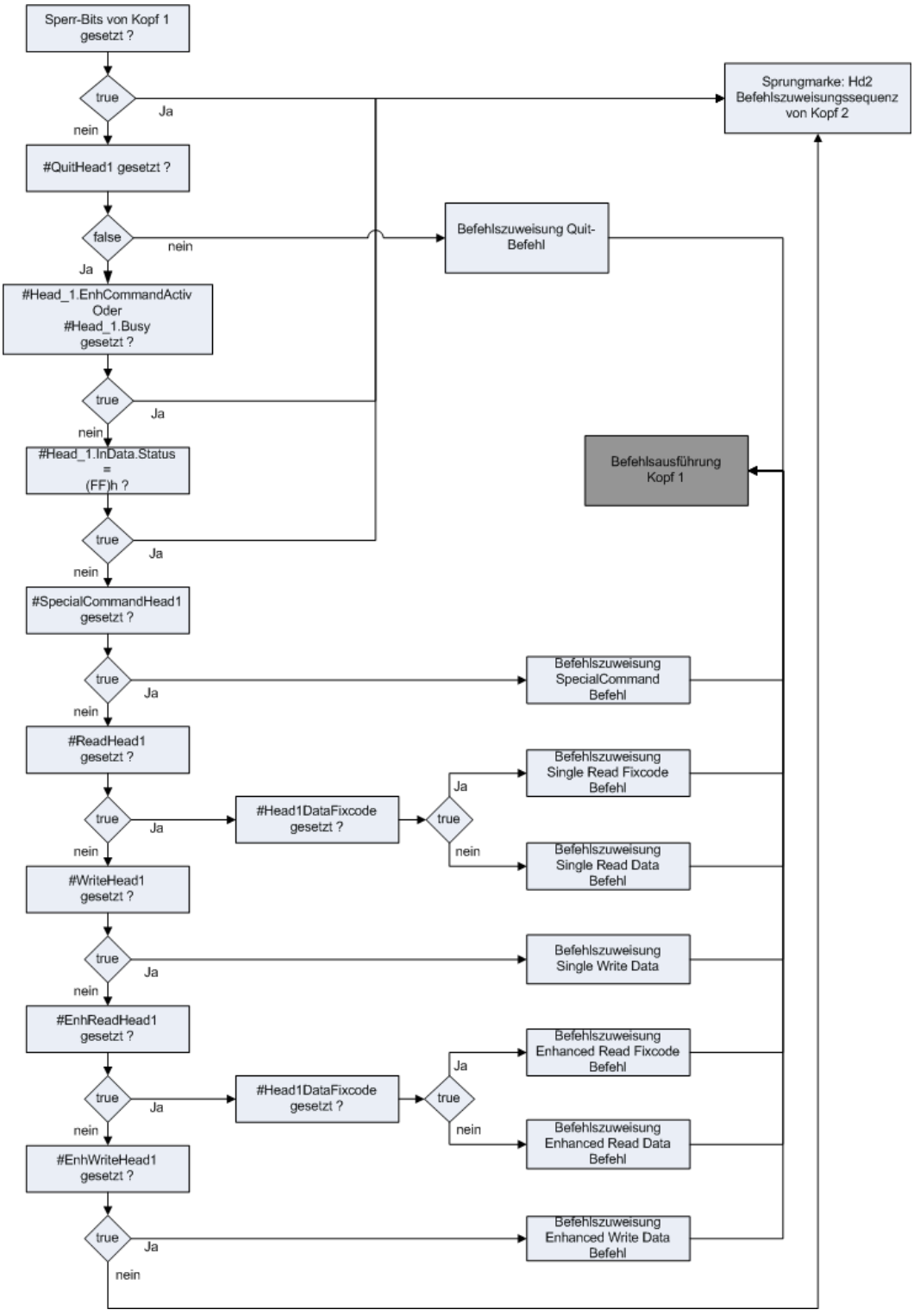

Abb. 14: Ablaufplan der Befehlszuweisungssequenz von Kopf 1

In diesem Abschnitt wird die Befehlszuweisung beispielhaft für Kopf 1 erläutert. Die Befehlszuweisungen der anderen Köpfe verhalten sich analog zu der von Kopf 1. Es

wird an dieser Stelle darauf verzichtet, näher auf die Befehlszuweisungssequenzen der Köpfe 2, 3 und 4 einzugehen.

Zu Beginn der Befehlszuweisung wird überprüft, ob der Kopf für eine Befehlsausführung blockiert ist. Es gibt verschiedene Möglichkeiten, die zur Blockierung der Befehlszuweisung führen. Ist bei vorangegangenen Befehlsausführungen an Kopf 1 ein Fehler aufgetreten, ist dieser Kopf für weitere Befehle gesperrt, bis er durch Setzen des Bits #Head_1.QuittError wieder in den Grundzustand versetzt wird. Eine Befehlszuweisung wird ebenfalls unterbunden, wenn kein Schreib-/Lesekopf am entsprechenden IDENT-Kanal angeschlossen ist. Die Befehlszuweisung ist aber ebenfalls blockiert, wenn sich schon Daten in dem Ausgangsdatenfeld befinden. Diese Daten stammen aus einer vorhergehenden Befehlszuweisung und wurden noch nicht an den entsprechenden Kopf versandt. Damit die Ausführung des in diesen Daten enthaltenen Befehls durchgeführt werden kann, ist die Befehlszuweisung für das entsprechende Ausgangsdatenfeld gesperrt bis der Befehl ausgeführt wurde. Die Abfrage wird mit einer Oder-Bedingung realisiert. Sobald eine der Bedingungen erfüllt ist, wird zur Befehlszuweisung von Kopf 2 an die Sprungmarke Hd2 gesprungen.

Im Anschluss an die Überprüfung auf eine Blockierung erfolgt die Zuweisung der Befehlsparameter in das Ausgangsdatenfeld. Dazu muss vorher ermittelt werden, welcher Befehl an Kopf 1 ausgeführt werden soll.

Als erstes wird überprüft, ob ein Quit-Befehl an Kopf 1 ausgeführt werden soll. Dazu wird das Bit #QuitHead1 abgefragt. Wenn kein Quit-Befehl gesetzt ist, so ist das Bit nicht gesetzt. Es erfolgt ein Sprung zur Sprungmarke SCH1. An dieser Sprungmarke erfolgt die Abfrage, ob ein Enhanced-Befehl aktiv ist.

Wenn ein Quit-Befehl ausgeführt werden soll, so ist das Bit #QuitHead1 gesetzt und es müssen nun die Befehlsparameter in das Ausgangsdatenfeld für Kopf 1 geladen werden. Nach der Zuweisung der Befehlsparameter in das Ausgangsdatenfeld werden verschiedene Sperrbits gesetzt. Durch das Bit #TransfToHeadX wird signalisiert, dass an Kopf X Befehlsparameter in das Ausgangsdatenfeld geladen wurden. Das Senden des Befehles an die IDENTControl hat aber noch nicht stattgefunden. Damit die Befehlsparameter des Quit-Befehls nach der ersten Befehlsausführung nicht nochmalig in das Ausgangsdatenfeld übergeben werden, setzt man das Bit #QuitHead1 zurück. Dadurch können andere Befehle an Kopf 1 wieder gestartet werden, sobald der Quit-Befehl ausgeführt wurde. Mit Hilfe des Rücksetzens der Bits #Head_1.EnhCommandActive und #Head_1.Busy wird deutlich, dass der Quit-Befehl

nur einmalig und nicht dauerhaft ausgeführt wird. Im Anschluss an die Parameterzuweisung für den Quit-Befehl in das Ausgangsdatenfeld erfolgt ein Sprung zur Befehlsausführung von Kopf 1 an der Sprungmarke exe1.

Wenn an Kopf 1 kein Quit-Befehl ausgeführt werden soll, so erfolgt ein Sprung zur Sprungmarke SCH1. Hier wird zunächst überprüft, ob ein Enhanced-Befehl an Kopf 1 ausgeführt wird oder ein anderer Befehl gerade in Bearbeitung ist. Wenn diese Bedingung erfüllt ist, wird der Kopf für die Ausführung weiterer Befehle gesperrt. Bei der Steuerung des Programms ist es während der Ausführung eines Enhanced-Befehls nicht möglich, einen anderen Befehl an Kopf 1 zu starten. Deshalb werden alle Startbit für die anderen Befehle zurückgesetzt. Eine Ausnahme bildet hier der Quit-Befehl. Dieser Befehl kann an dieser Stelle trotz aktiven Enhanced-Befehls zugewiesen und ausgeführt werden, weil er dadurch zum Abbruch des laufenden Enhanced-Befehls führt. Wenn ein Enhanced-Befehl aktiv ist, erfolgt im Anschluss an das Rücksetzen der Startbits ein Sprung zur Sprungmarke Hd2 zur Befehlszuweisung für Kopf 2. Anschließend wird überprüft, ob der Status der zuletzt an Kopf 1 eingelesenen Daten den Wert (FF)h aufweist und gleichzeitig ein Single-Befehl aktiv ist. In diesem Fall kann keine Befehlszuweisung durchgeführt werden und es erfolgt ein Sprung zur Befehlszuweisung von Kopf 2. Danach wird überprüft, welcher Befehl an Kopf 1durchgeführt werden soll.

Wenn die Ausführung eines Special-Commands an Kopf 1 beabsichtigt ist, wird dies durch das gesetzte Bit #SpecialCommandHead1 signalisiert. Daraufhin wird an die Sprungmarke SH1 gesprungen. An der Sprungmarke SH1 werden die durch den Anwender festgelegten Befehlsparameter in das Ausgangsdatenfeld von Kopf 1 kopiert.

Soll an Kopf 1 ein Read-Befehl gestartet werden, ist das Bit #ReadHead1 gesetzt und es wird zur Sprungmarke SRH1 gesprungen. An der Sprungmarke werden die Befehlsparameter zur Ausführung eines Read-Befehls an Kopf 1 in das zugehörige Ausgangsdatenfeld kopiert.

Bei einer beabsichtigten Ausführung eines Write-Befehls an Kopf 1 ist das Bit #WriteHead1 gesetzt. In diesem Fall erfolgt ein Sprung zur Sprungmarke SWH1. Hier wird analog zu den anderen Befehlen die Parameterübergabe in das Ausgangsdatenfeld für einen Single-Write-Befehl durchgeführt.

Bei der Ausführung eines Enhanced-Befehls ist zwischen einem Enhanced-Read und einem Enhanced-Write-Befehl zu unterscheiden. Zur Ausführung eines Enhan-

ced-Read-Befehls wird das Bit #EnhReadHead1 überprüft und es erfolgt ein Sprung zur Sprungmarke ERH1, sofern das Bit gesetzt ist. Andernfalls wird überprüft, ob ein Enhanced-Write-Befehl ausgeführt werden soll. Ist die Ausführung dieses Befehls beabsichtigt, so ist das Bit #EnhWriteHead1 gesetzt und es erfolgt ein Sprung zur Sprungmarke EWH1. Durch diese gestaffelte Abfrage werden alle möglichen, durch den Anwender startbaren, Befehle überprüft und in die entsprechende Befehlsparameterzuweisung gesprungen. Es besteht aber auch die Möglichkeit, keinen Befehl an Kopf 1 auszuführen. Dann sind alle der zu überprüfenden Bits nicht gesetzt und es erfolgt ein Sprung zu der Befehlszuweisung von Kopf 2 an die Sprungmarke Hd2. Nach der Zuweisung der Befehlsparameter in das Ausgangsdatenfeld wird zur Befehlsausführung von Kopf 1 an die Sprungmerke exe1 gesprungen. Dort wird das Ausgangsdatenfeld von Kopf 1 mit Hilfe der Funktion SFC 15 an die IDENTControl übertragen.

2.4.7 Ablauf der Befehlsausführung für Kopf 1

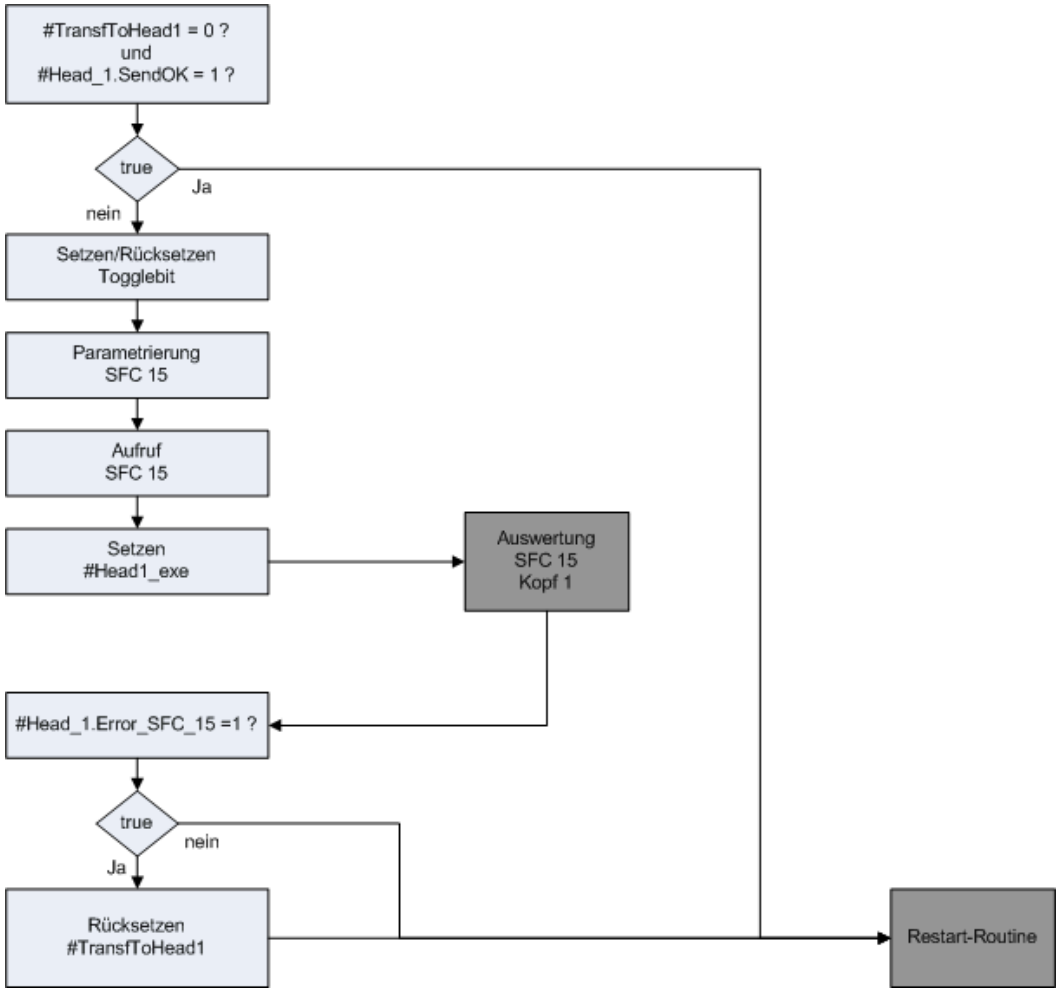

Abb. 15 Ablaufplan Befehlsausführung von Kopf 1

In diesem Abschnitt wird beispielhaft der Ablauf der Befehlsausführung für Kopf 1 erläutert. Die Bearbeitung der Befehlsausführung für die Köpfe 2, 3 und 4 verhält sich analog. Deshalb wird an dieser Stelle darauf verzichtet, näher auf die Befehlsausführung dieser Köpfe einzugehen.

Unter der Befehlsausführung von Kopf 1 versteht man das Senden der in dem zugehörigen Ausgangsdatenfeld abgelegten Befehlsparameter an die IDENTControl. Die Zuweisung der Befehlsparameter in die zugehörigen Ausgangsdatenfelder wurde im vorhergehenden Abschnitt erläutert. Das Senden der Ausgangsdatenfelder wird durch die Systemfunktion SFC 15 durchgeführt. Nach erfolgreicher Übermittelung des Ausgangsdatenfeldes wird der Befehl über eine Statusrückmeldung von der IDENTControl quittiert und ausgeführt.

An der Sprungmarke exe1 wird zunächst überprüft, ob das Bit #TransfToHead1 nicht gesetzt und das Bit #Head_1.SendOK gesetzt ist. In diesem Fall ist bereits ein Befehl an Kopf 1 gesendet wurden und es wird zur Restart-Routine gesprungen.
Anschließend erfolgt das Setzen bzw. Rücksetzen des Togglebits. Dies ist erforderlich um aufeinander folgende identische Befehle voneinander unterscheiden zu können.
Im nächsten Schritt werden die verschiedenen Sperrbits gesetzt und die Parametrierung der Funktion SFC 15 durchgeführt. Als Quelldatenfeld für den Aufruf der Funktion SFC 15 dient das Ausgangsdatenfeld für Kopf 1 #Head_1.OutData. An die Parametrierung schließt sich der Aufruf der Funktion SFC 15 an. Die Variable #Head1_exe dient zur Unterscheidung des Aufrufes der Funktion FC 50. In dem Funktionsbaustein sind zwei mögliche Stellen vorhanden, an denen die Funktion für Kopf 1 aufgerufen wird. Dies ist zum einen die Initialisierung des ersten Kopfes und die andere Stelle ist die Befehlsausführung von Kopf 1. Beim Aufruf der Funktion in der Initialisierungsphase wird das Bit #Head1_exe zurückgesetzt. Hingegen wird das Bit gesetzt, wenn der Aufruf bei der Befehlsausführung erfolgt. Somit kann abhängig vom Ort des Funktionsaufrufes ein zielgerichteter Sprung realisiert werden.
Nach dem Setzen von #Head1_exe wird zur Sprungmarke aus1 in das Netzwerk 19 gesprungen. Dort wird überprüft, ob bei der Ausführung der Funktion SFC 15 ein Fehler aufgetreten ist. Über die Variable #Head1_exe wird der Rücksprung zur Sprungmarke F155 realisiert. An dieser Stelle wird das Bit #TransfToHead1 zurückgesetzt, wenn bei der Ausführung der Funktion SFC 15 ein Fehler aufgetreten ist. Abschließend erfolgt ein Sprung zur Restart-Routine.

2.4.8 Ablauf der Restart-Routine

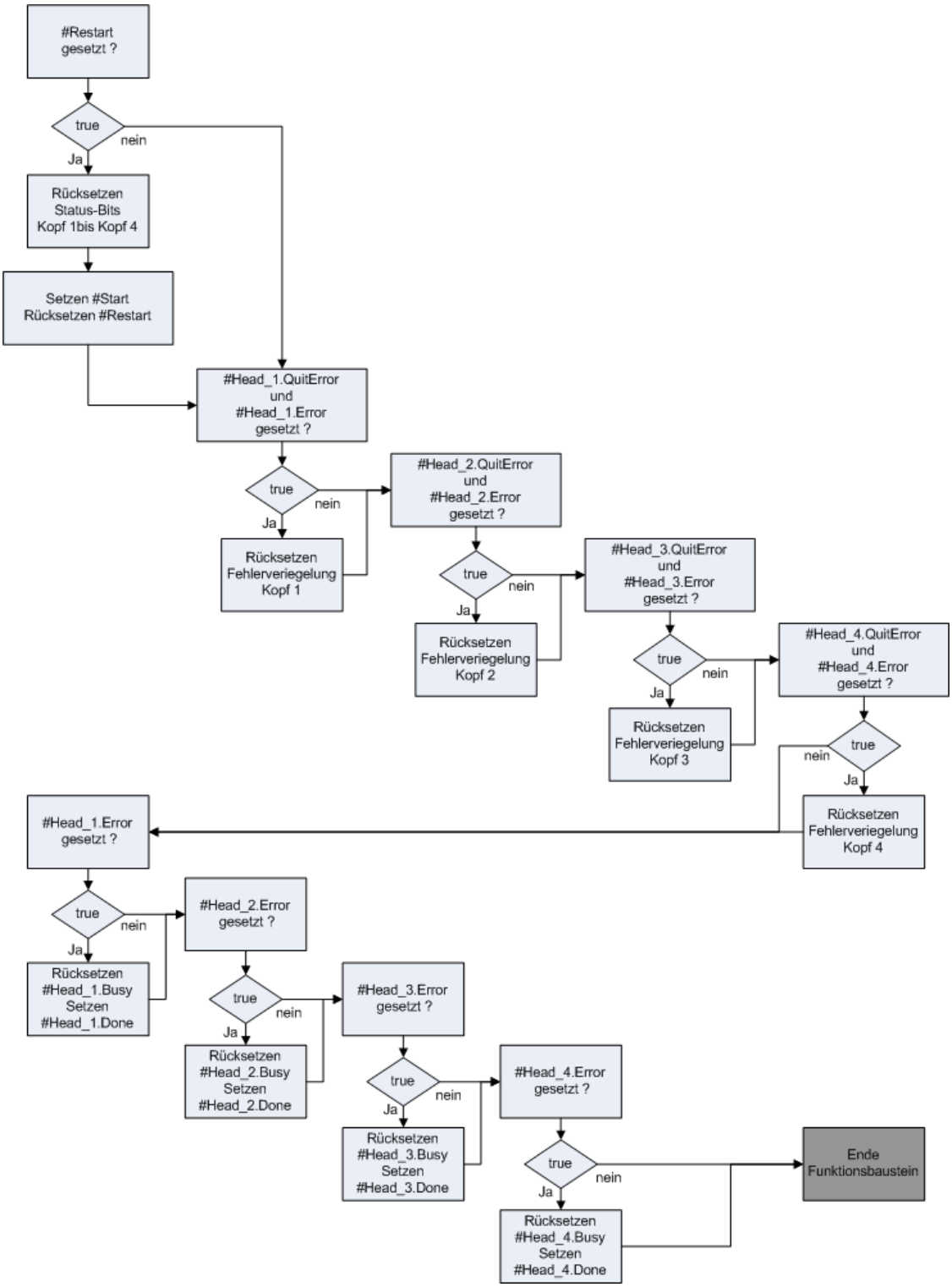

Abb. 16: Ablaufplan Restart-Routine

Die Restart-Routine hat die Aufgabe den Funktionsbaustein in einen definierten Grundzustand zurückzusetzen und zu beenden.

Nachfolgend ist die Restart-Routine beispielhaft für den Kopf 1 beschrieben. Die Köpfe 2, 3 und 4 verhalten sich analog zu dem Beispiel.

Zu Beginn der Restart-Routine wird überprüft, ob das Bit #Restart gesetzt ist. Das Bit ist gesetzt, wenn der Funktionsbaustein einen Restart vornehmen soll. Sollte das Bit aber nicht gesetzt sein, so erfolgt ein Sprung zur Sprungmarke end2. Die Sprungmarke end2 befindet sich am Anfang des Netzwerkes 18. Dort erfolgt das Rücksetzen Bits, die einen Fehlerzustand melden.

Wenn das Bit #Restart gesetzt ist, dann werden die für den Kopf 1 erforderlichen Zustandsbits zurückgesetzt.

Das Rücksetzen der Zustandsbits für die Köpfe 2, 3 und 4 schließt sich direkt an den beispielhaft gezeigten Ausschnitt der Restart-Routine an.

Nach dem Rücksetzen der Zustandsbits für alle Köpfe wird das Bit #Start gesetzt. Durch dieses Bit wird im nächsten Durchlauf des Funktionsbausteins die Initialisierung aller Köpfe innerhalb der Start-Up-Sequence in Netzwerk 1 überprüft. Durch das Rücksetzen des Bits #Restart wird signalisiert, das der Funktionsbaustein in einen definierten Grundzustand zurückversetzt wurde.

Ein Bestandteil der Restart-Routine ist die Quittierung von Fehlermeldebits. Dieser Pogrammteil wird durchlaufen wenn zu Beginn der Restart-Routine das Bit #Restart nicht gesetzt ist. Dann wird direkt zu der Quittierung der Fehlermeldungen gesprungen. Die Quittierung wird ebenfalls durchlaufen wenn ein Restart erfolgt ist. Sobald bei der Ausführung des Funktionsbausteins an einen Kopf ein Fehler aufgetreten ist, wird das entsprechende Fehlermeldebit sowie das Bit #Head_X.Error gesetzt. Dadurch wird die Ausführung weiterer Befehle an diesen Kopf gesperrt. Der Funktionsbaustein bietet den Anwender nun die Möglichkeit die Fehlerverriegelung des entsprechenden Kopfes aufzuheben. Dazu muss die IN-Variable #QuitErrorHeadX durch den Anwender gesetzt werden. In der Variablenumsetzung wird diese IN-Variable auf die STAT-Variable #Head_X.QuitError umgesetzt.

Am Anfang der Quittierungssequenz wird zunächst überprüft, ob die Bits #Head_1.QuitError und #Head_1.Error gesetzt sind. Wenn die Bedingung erfüllt ist, so ist bei der Bearbeitung des Funktionsbausteins ein Fehler an Kopf 1 aufgetreten und der Anwender möchte die Fehlerverriegelung aufheben. Daraufhin werden die entsprechenden Fehlermelde- und Statusbits zurückgesetzt.

Die Quittierung der Fehlerverriegelung für die übrigen Köpfe schließt sich direkt an die beispielhaft gezeigte Quittierungssequenz und erfolgt analog. An dieser Stelle wird darauf verzichtet näher auf die Quittierungssequenzen der übrigen Köpfe einzugehen.

Nach der Quittierungssequenz erfolgt ein Programmteil zur Bearbeitung von Fehlermeldungen. Die Quittierungssequenz wird nur durchlaufen, wenn der Anwender die Fehlerverriegelung des entsprechenden Kopfes aufheben will. Wenn die Fehlerverriegelung nicht aufgehoben wurde, so müssen Statusbits gesetzt werden. Diese Statusbits haben die Aufgabe zu signalisieren, dass der Befehl wegen eines Fehlers beendet wurde.

Zu Beginn wird überprüft, ob ein Fehler aufgetreten ist. Dazu wird das Bit #Head_1.Error geprüft. Wenn das Bit gesetzt ist, so wird das Bit #Head_1.Busy zurückgesetzt. Dieses Bit signalisiert die laufende Ausführung eines Befehles. Anschließend wird das Bit #Head_1.Done gesetzt. Dadurch wird signalisiert, dass die Bearbeitung des Befehles beendet ist.

Das Rücksetzen der Statusbits wird nacheinander für alle weiteren Köpfe durchgeführt. Danach erfolgt ein Sprung zur Sprungmarke endG. Die Sprungmarke endG stellt das Ende des Funktionsbausteins dar.

2.4.9 Ablauf der Analyse der Funktion SFC 15

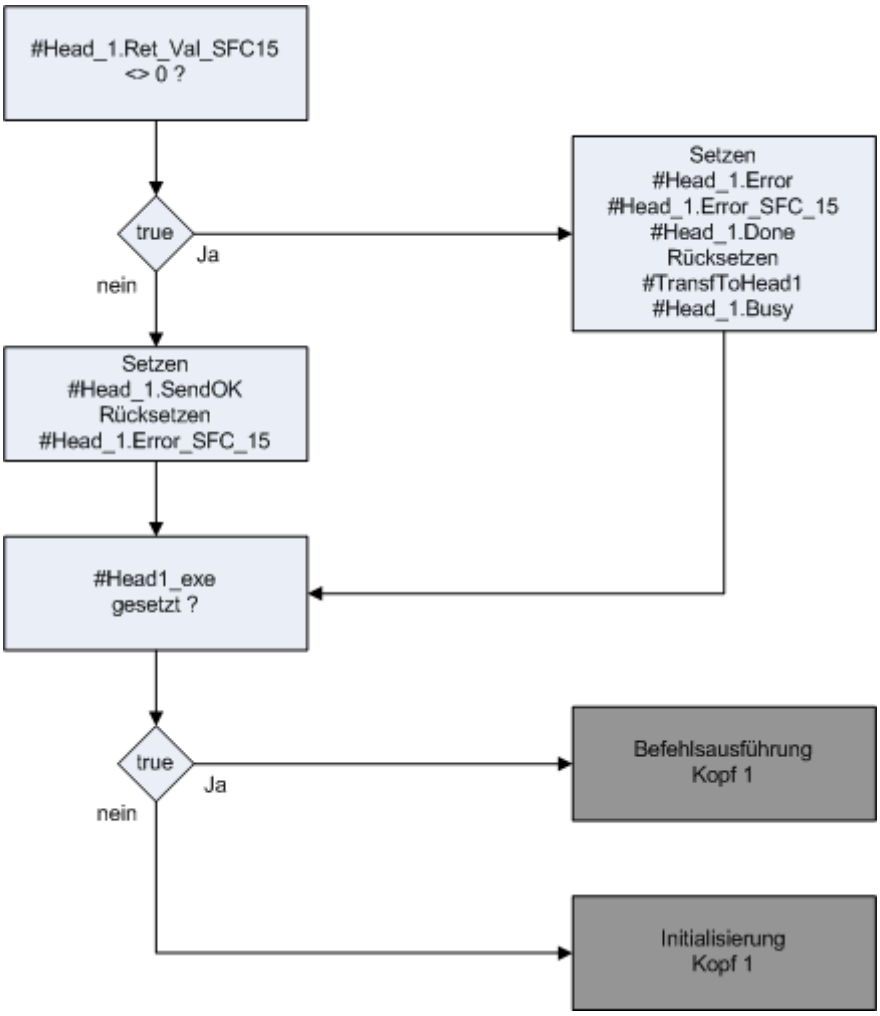

Abb. 17: Ablaufplan Analyse SFC 15 Kopf 1

Die Analyse der Funktion SFC 15 hat die Aufgabe, einen Fehler bei der Ausführung der Funktion SFC 15 festzustellen. Die Funktion SFC 15 wird innerhalb des Funktionsbausteins bei der Initialisierung und dem Ausführungsteil der Befehle aufgerufen. Nach der Ausführung der Funktion in den entsprechenden Programmteilen, erfolgt ein Sprung zur Analyse des SFC 15. Mit Hilfe der Analyse wird festgestellt, ob bei der Ausführung der Funktion ein Fehler aufgetreten ist und die Daten der Ausgangsdatenfelder nicht an die IDENTControl übermittelt werden konnten.

Nachfolgend ist der Ablauf der Analyse der Funktion SFC 15 beispielhaft für den Kopf 1 erläutert. Für die Köpfe 2, 3 und 4 erfolgt der Ablauf der Analyse analog. Deshalb wird an dieser Stelle darauf verzichtet, näher auf die übrigen Analysen einzugehen.

Zu Beginn der Analyse wird zunächst überprüft, ob die Variable #Head_1.Ret_Val_SFC15 mit einem Wert ungleich 0 belegt ist. In dieser Variablen

ist ein Fehlercode enthalten, wenn bei der Ausführung der Funktion SFC 15 ein Fehler aufgetreten ist. In diesem Falle wird zur Sprungmarke e151 gesprungen. Dort werden die Fehlerbits gesetzt, die einen Fehler bei der Ausführung der Funktion SFC 15 an Kopf 1 signalisieren.

Ist bei der Ausführung der SFC 15 kein Fehler aufgetreten, wird das Bit #Head_1.SendOK gesetzt und zur Sprungmarke spr1 gesprungen.

An dieser Sprungmarke wird der Rücksprung aus der Analyse der Funktion SFC 15 mit Hilfe des Bits #Head1_exe realisiert. Ist das Bit gesetzt, erfolgt der Rücksprung zur Sprungmarke F155 zur Befehlsausführung von Kopf 1. Andernfalls zur Sprungmarke F151 zur Initialisierung von Kopf 1.

2.4.10 Ablauf der Analyse der Funktion SFC 14

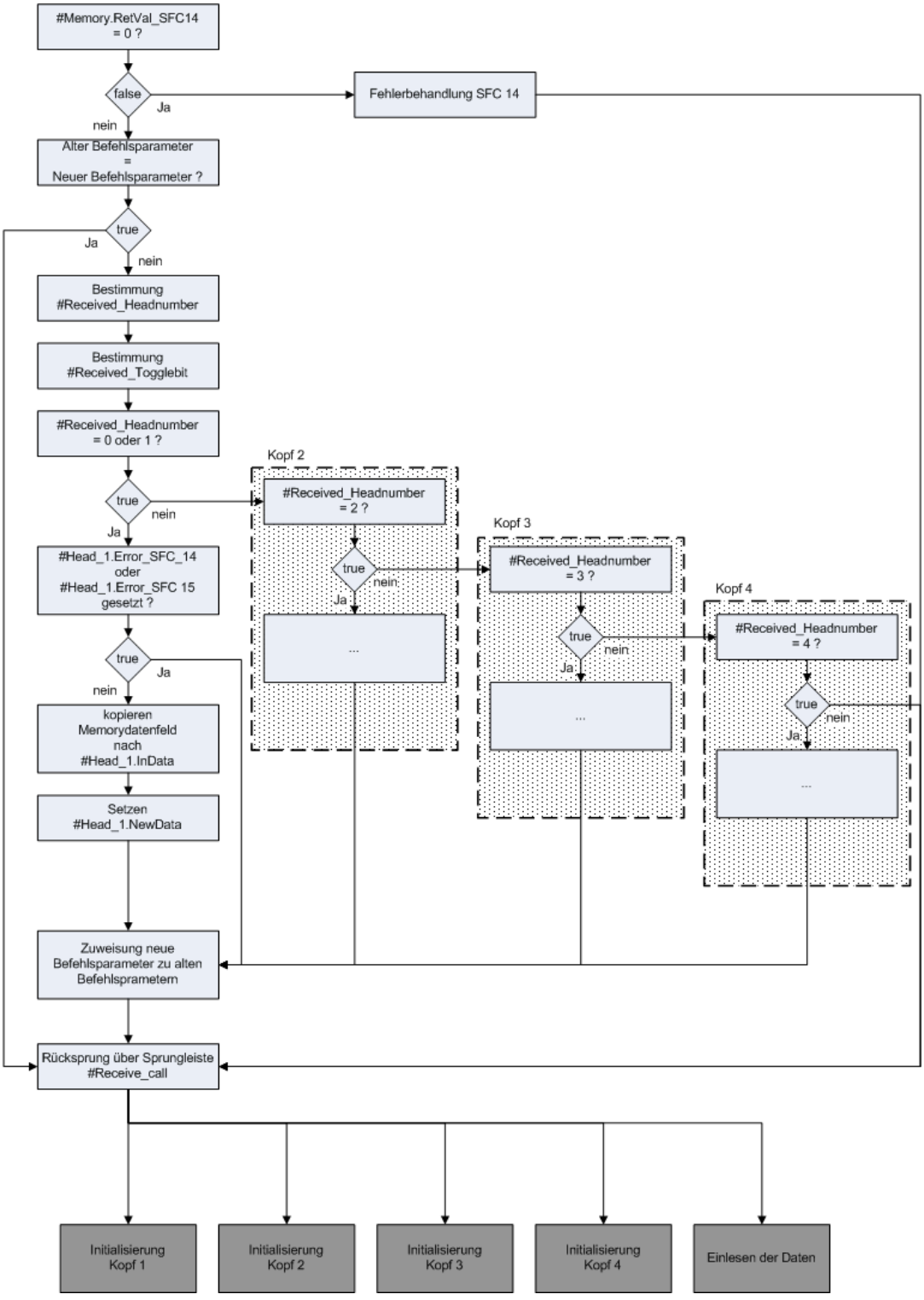

Abb. 18: Ablaufplan Analyse SFC 14

Die Analyse der Funktion SFC 14 hat die Aufgabe, einen Fehler bei der Ausführung der Funktion SFC 14 festzustellen. Die Funktion SFC 14 wird innerhalb des Funktionsbausteins bei der Initialisierung und bei der Einlese-Routine aufgerufen. Nach der

Ausführung der Funktion SFC 14 in den entsprechenden Programmabschnitten, erfolgt ein Sprung zur Analyse der Funktion SFC 14 in das Netzwerk 20. Zunächst wird überprüft, ob bei der Ausführung der Funktion ein Fehler aufgetreten ist. Dazu wird überprüft, ob die Variable #Memory.RetVal_SFC14 mit einen Wert belegt ist. In diesem Fall erfolgt eine entsprechende Fehlerbehandlung an der Sprungmarke er14. Wenn die Funktion SFC 14 fehlerfrei ausgeführt werden konnte, so werden zunächst die neu eingelesenen Befehlsparameter mit den vorhergehenden Befehlsparametern verglichen. Stimmen diese nicht überein, so wurden neue Daten eingelesen. Wenn neue Daten eingelesen wurden, wird zunächst die Kopfnummer der eingelesenen Daten aus den Memorydatenfeld isoliert. Die Kopfnummer der eingelesenen Daten wird der Variablen #Received_Headnumber zugewiesen. Im nächsten Schritt wird das Togglebit bestimmt und der Variablen #Received_Togglebit zugewiesen.

Anschließend wird anhand der Kopfnummer das Memorydatenfeld in das zugehörige Eingangsdatenfeld kopiert. Daraufhin wird das Bit #Head_X.NewData gesetzt. Dieses Bit signalisiert, dass neue Daten im Eingangsdatenfeld des zugehörigen Kopfes zur Auswertung bereitstehen. Danach werden die neu eingelesenen Befehlsparameter den alten Parametern zugewiesen.

Im letzten Schritt erfolgt mit Hilfe einer Sprungleiste der Rücksprung in das Programmteil, von dem aus die Analyse aufgerufen wurde.

2.4.11 Analyse der Eingangsdatenfelder

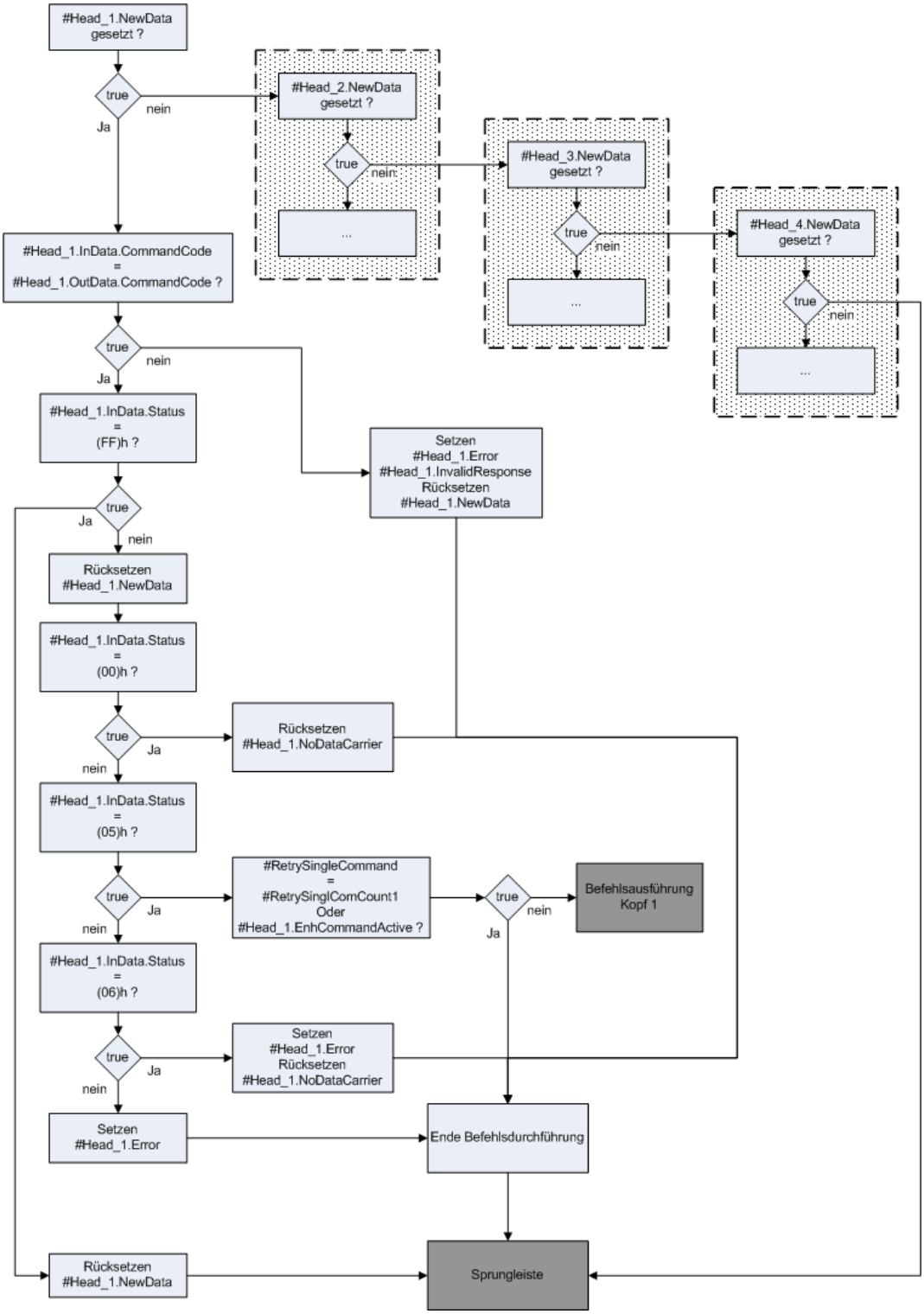

Abb. 19: Ablaufplan Analyse Eingangsdatenfelder

Dieser Programmteil des Funktionsbausteins hat die Aufgabe die neu eingelesenen Daten der Eingangsdatenfelder zu analysieren. Die Analyse der Eingangsdatenfelder

ist erforderlich sobald Daten aus den Memorydatenfeld in die jeweiligen Eingangsdatenfelder kopiert wurden.

Zu Beginn der Analyse wird zunächst festgestellt, in welchen der Eingangsdatenfelder neue Daten vorhanden sind. Wenn in einen Eingangsdatenfeld neue Daten zur Auswertung bereitstehen, so ist das Bit #Head_1.NewData gesetzt und es wird zur Auswertung an die entsprechende Sprungmarke gesprungen.

Dort erfolgt die eigentliche Analyse der Eingangsdaten. Dazu wird zunächst der Befehlscode des Ausgangsdatenfeldes mit dem Befehlscode des Eingangsdatenfeldes verglichen. Sind die beiden Befehlscodes nicht identisch, so ist bei der Befehlsausführung der IDENTControl ein Fehler aufgetreten und es erfolgt ein Sprung zur Sprungmarke err1. Dort werden die Fehlermeldebit #Head_1.Error und #Head_1.InvalidResponse gesetzt.

Wenn beide Befehlscodes gleich sind, so wird im nächsten Schritt der Status der eingelesenen Daten überprüft.

Hat der Status den Wert (FF)h, so wurde der an die IDENTControl gesendete Befehl zwar „verstanden“, aber der Befehl wird zurzeit noch ausgeführt, so dass die eingelesenen Daten verworfen werden müssen. Deshalb wird an der Sprungmarke sff1 das Bit #Head_1.NewData zurückgesetzt.

Wenn die Befehlsdurchführung der IDENTControl abgeschlossen ist, so hat der Status der Wert (00)h.

Der Status der eingelesenen Daten hat den Wert (05)h, wenn bei der Befehlsausführung kein Datenträger sich im Erfassungsbereich des Schreiblesekopfes befunden hat. Dadurch wird der Ausführungszähler inkrementiert und der Befehl nochmals ausgeführt. Wenn die maximale Anzahl der Befehlswiederholungen erreicht ist, so wird das Bit #Head_1.NoDataCarrier gesetzt und die Befehlsausführung wird nicht wiederholt.

Besitzen die Eingangsdaten einen anderen Statuswert, so ist ein Fehler bei der Befehlsdurchführung aufgetreten und es wird das Fehlermeldebit #Head_1.Error gesetzt.

Abschließend wird über eine Sprungleiste zum Ausgangspunkt der Analyse der Eingangsdatenfelder gesprungen.

3. Anhang

3.1 Auflistung der Parameter

3.1.1 Eingangsparameter (IN-Parameter)

- IDENT_Control_Address
  Adresse der IDENTControl innerhalb des Profibus-Netzwerks. Die Angabe bezieht sich auf die E/A-Adressen der IDENTControl aus der Hardware-Konfiguration
- Timeout
  Zeitfenster zur Timeoutüberwachung
- RetrySingleCommand
  Anzahl der maximalen Befehlswiderholungen
- HeadXDataFixcode
  HeadXDataFixcode = 0 **à** Zugriff auf den Datenbereich
  HeadXDataFixcode = 1 **à** Zugriff auf Fixcode
- HeadXSingleEnhanced
  HeadXSingleEnhanced = 0 **à** Ausführung eines Single-Befehls
  HeadXSingleEnhanced = 1 **à** Ausführung eines Enhanced-Befehls
- HeadXQuit
  Start eines Quit-Befehls zum Abbruch eines Enhanced-Befehls
- HeadXRead
  Start eines Read-Befehls
- HeadXWrite
  Start eines Write-Befehls
- QuitErrorHeadX
  Start eines QuitError-Befehls zur Aufhebung der Fehlerverriegelung
- HeadXSpecialCommand
  Start eines Special-Commands
- IC_Command_on_Head1
  Start eines Befehls an die IDENTControl. Dabei erfolgt keine Angabe des Kanals und somit keine Befehlsausführung durch die Schreib-/Leseköpfe

3.1.2 Durchgangsparameter (IN-OUT-Parameter)

- InitFinish
  Ende der Initialisierung aller Köpfe
- SetRestart
  Start eines Restarts

3.1.3 Statische Parameter (STAT-Parameter)

- Byte_number
  Anzahl der Telegrammgröße (z.B. Byte_number = 32 bei „IN/Out 32 Byte“
- Head_X.InData
  Head_X.InData ist eine Datenstruktur, welche die Rückantwort der IDENTControl auf einen Befehl enthält. Die Datenstruktur setzt sich aus verschiedenen Elementen unterschiedlichen Datentyps zusammen.
  - Head_X.InData.CommandCode
    Befehlscode der eingelesenen Antwort
  - Head_X.InData.Channel
    Obere 4 Bits enthalten mögliche Wortanzahl; untere 4 Bits enthalten die Kanalkennung und das Togglebit
  - Head_X.InData.Status

Ausführungszustand
  - Head_X.InData.ExecutionCounter
    Ausführungszähler
  - Head_X.InData.DW1 … DW15
    Datenfeld für die eingelesenen Nutzdatenwörter
- Head_X.OutData
  Head_X.OutData ist ebenfalls eine Datenstruktur. Diese Struktur beinhaltet die Daten, welche zur Ausführung eines Befehls an die IDENTControl gesendet werden. Das Datenfeld ist in mehrere Elemente unterschiedlichen Datentyps gegliedert.
  - Head_X.OutData.CommandCode
    Befehlscode des auszuführenden Befehls
  - Head_X.OutData.Channel
    Kanalkennung
  - Head_X.OutData.Wordadr_High
    Oberes Byte der Adresse, ab der die Daten gelesen/geschrieben werden; bei Change-Tag Befehl **à** oberes Byte der Datenträgerkennung
  - Head_X.OutData.Wordadr_Low
    unteres Byte der Adresse, ab der die Daten gelesen/geschrieben werden; bei Change-Tag Befehl **à** unteres Byte der Datenträgerkennung
  - Head_X.OutData.DW1 … DW15
    Ausgangsdatenfeld der Nutzdatenwörter
- Head_X.WordAddress
  Startadresse für Datenträgerzugriff
- Head_X.TimeoutActiv
  Timeoutüberwachung ist aktiv
- Head_X.InvalidResponse
  Ungültige Antwort der IDENTControl
- Head_X.QuitError
  Aufheben der Fehlerverriegelung
- Head_X.NewData
  Neue Daten stehen zur Analyse bereit
- Head_X.NotExist
  Kein Schreib-/Lesekopf angeschlossen
- Head_X.ExistTC
  Schreib-/Lesekopf angeschlossen
- Head_X.Error
  Fehler in der Befehlsausführung der IDENTControl
- Head_X.TimeoutOccured
  Zeitfenster der Timeoutüberwachung abgelaufen
- Head_X.ReceiveOK
  Antwort der IDENTControl empfangen
- Head_X.SendOK
  Daten an die IDENTControl gesendet
- Head_X.NoDataCarrier
  Kein Datenträger im Erfassungsbereich
- Head_X.Done
  Enhanced-Befehl **à** Daten gelesen bzw. geschrieben
  Single-Befehl **à** Befehlsausführung beendet
- Head_X.Busy
  Befehl in Bearbeitung
- Head_X.Error_SFC_14
  Fehler bei der Ausführung der SFC 14
- Head_X.Error_SFC_15
  Fehler bei der Ausführung der SFC 15
- Head_X.EnhCommandActive

Enhanced-Befehl wird ausgeführt

- Head_X.SglCommandActive
  Single-Befehl wird ausgeführt
- Head_X.WordNum
  Anzahl der zu übertragenden Nutzdatenwörter
- Head_X.RetVal_SFC14
  Enthält einen Fehlercode bei der Ausführung der SFC 14
- Head_X.RetVal_SFC15
  Enthält einen Fehlercode bei der Ausführung der SFC 15
- Head_X.SpecialCommand
  Das Datenfeld enthält die Parameter zur Ausführung eines Special-Commands. Mit Hilfe eines Special-Commands kann man Befehle ausführen, die nicht durch die Befehlsliste des Funktionsbausteins abgedeckt sind. Zur Ausführung eines Special-Commands muss der Anwender die Befehlsparameter des auszuführenden Befehls in dieses Datenfeld eingeben. Die hier eingetragenen Parameter werden dann innerhalb des Funktionsbausteins in das Ausgangsdatenfeld überführt und zur IDENTControl übertragen.
  - Head_X.SpecialCommand.Code
    Befehlscode
  - Head_X.SpecialCommand.Channel
    Kanalkennung und evtl. Anzahl zu übertragender Nutzdatenwörter
  - Head_X.SpecialCommand.Parameter1 …5
    Befehlsparameter
- Head_X.Memory
  Das Memory-Datenfeld enthält die Daten, welche von der IDENTControl an die SPS gesendet werden. Alle eingehenden Daten werden unabhängig vom IDENT-Kanal in diesem Datenfeld abgelegt. Nach der Überprüfung des IDENT-Kanals der eingehenden Daten, erfolgt die Umkopierung des Memory-Datenfeldes in das Eingangsdatenfeld des entsprechenden IDENT-Kanals. Der Aufbau des Memory-Datenfeldes ist identisch zu den Eingangsdatenfeldern der IDENT-Kanäle und wird deshalb hier nicht näher beschrieben.

3.2 Befehlsliste

BEFEHL	CODE	PARAMETRIERUNG	AUSFÜHRUNG
Single-Read-Fixcode	(01)h	Keine	#HeadXDataFixcode = 1 #HeadXRead = 1 #HeadXWrite = 0 #HeadXSingleEnhanced = 0
Enhanced-Read-Fixcode	(1D)h	Keine	#HeadXDataFixcode = 1 #HeadXRead = 1 #HeadXWrite = 0 #HeadXSingleEnhanced = 1
Single-Read-Data	(10)h	#HeadX.WordAddress #HeadX.WordNum	#HeadXDataFixcode = 0 #HeadXRead = 1 #HeadXWrite = 0 #HeadXSingleEnhanced = 0
Enhanced-Read-Data	(19)h	#HeadX.WordAddress #HeadX.WordNum	#HeadXDataFixcode = 0 #HeadXRead = 1 #HeadXWrite = 0 #HeadXSingleEnhanced = 1
Single-Write-Data	(10)h	#HeadX.WordAddress #HeadX.WordNum #HeadX.OutData.DW	#HeadXDataFixcode = 0 #HeadXRead = 0 #HeadXWrite = 1 #HeadXSingleEnhanced = 0
Enhanced-Write-Data	(19)h	#HeadXWordAddress #HeadXWordNum #HeadX.OutData.DW	#HeadXDataFixcode = 0 #HeadXRead = 0 #HeadXWrite = 1 #HeadXSingleEnhanced = 0

Special-Command	(??)h	#Head_X.SpecialCommand.Code #Head_X.SpecialCommand.Channel #Head_X.SpecialCommand.Parameter	#HeadXSpecialCommand = 1 #IC_Command_on_Head1 = 0
IDENT-Control-Command	(??)h	#Head_1.SpecialCommand.Code #Head_1.SpecialCommand.Parameter	#Head1SpecialCommand = 1 #IC_Command_on_Head1 = 1
QuitError-Befehl	-	Keine	#QuitErrorHeadX
Quit-Befehl	-	Keine	#HeadXQuit

3.3 Code-/Datenträger

TYP	#HEADXTAGTYPE	ZUGRIFF	DATENBEREICH	FIXCODELÄNGE
IPC02-..	W#16#3032	Read Fixcode	-	5 Byte
IPC03-..	W#16#3033	Read Fixcode Read Data Write Data	116 Byte	4 Byte
IPC10-..	W#16#3130	Read Data Write Data	12 Byte	-
IPC11-..	W#16#3131	Read Data Write Data	5 Byte	-
IPC12-..	W#16#3132	Read Fixcode Read Data Write Data	8 kByte	4 Byte
IPC14-..	W#16#3134	Read Data Write Data	5 Byte	-
IQC20-..	W#16#3230	Read Fixcode Read Data Write Data		8 Byte
IQC21-..	W#16#3231	Read Fixcode Read Data Write Data	112 Byte	8 Byte
IQC22-..	W#16#3232	Read Fixcode Read Data Write Data	256 Byte	8 Byte
IDC-…1k	W#16#3530	Read Fixcode Read Data Write Data	128 Byte	4 Byte
ICC-..	W#16#3532	Read Fixcode	-	7 Byte
MVC-60	W#16#3630	Read Fixcode Read Data Write Data	8 kByte	-