ADDI-DATA®

Technical support:
+49 (0)7223 / 9493-0

Preliminary version

Technical documentation

ADDICOUNT APCI-/CPCI-1710

Synchronous Serial Interface

2nd edition 01/1999

Tables

Figures

1 INTRODUCTION

1.1 Technical documentation

This handbook refers to the **APCI-1710** and **CPCI-1710 SSI** boards. Make sure that you have received the following items:

- the handbook **ADDICOUNT APCI-/CPCI-1710: Function-programmable counter board for the PCI bus**, which contains general information on the operation of the board
- the yellow leaflet "Safety precautions"

Please note:

The APCI-1710 board is compatible to the CPCI-1710 board as far as the installation of the software is concerned. The ADDIREG and SET1710 programs make no difference between PCI and CompactPCI boards.

1.2 Function description

Apart from a global description of the functions this handbook contains:

- the pin assignment of the front connector
- a list of the signals used
- the I/O mapping
- connection examples
- a chapter on the API software functions of the supplied device driver

2 SYNCHRONOUS SERIAL INTERFACE (SSI)

2.1 Function description

The SSI function is an interface for absolute angle encoders (displacement measurement systems).
It allows the transmission of an absolute position information through a serial data transfer.
It stands out for applications in which high precision and reliability are necessary in a rough industrial environment.

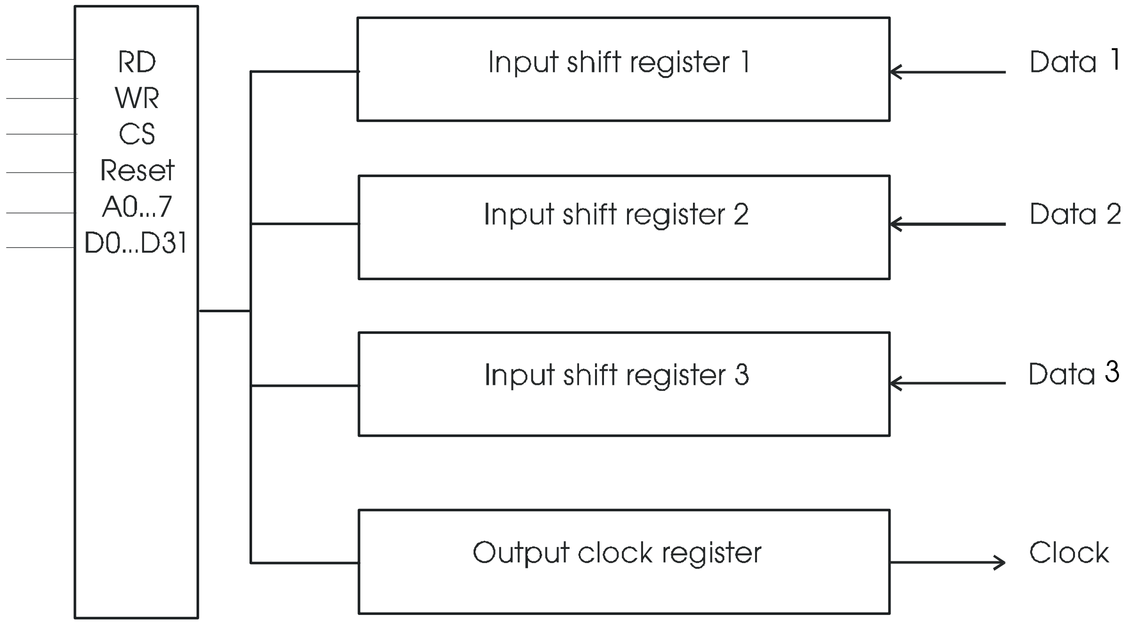
Advantages in comparison with the parallel interface:
- cable length requirement is higly reduced
- The requirement of cabling and interface technology does not depend on the data word length
- The shielding capacity against interferences is reached through synchronous and symmetric clock and data signals via a cable equipped with twisted pair lines.
- The optical isolation is entirely completed by optical couplers to avoid any earth cicuit.

Properties:
Connection of 1 to 3 encoders with SSI:
- The clock is common to all of the 3 encoders.
- The clock frequency is controlled through software to adapt the transfer of the control length.
- The number of data bits is programmable per software, which allows flexibility of the resolution.
- GRAY or BINARY conversions are possible
- 3 digital input and 1 digital output channels are available.
  These channels do no affect the SSI functions; they can bu used for an additional function.

2.1.1 Block diagram of the SSI encoder

The interface module contains:
- 3 independant 32-bit shift registers which are readable via the data bus
- clock and pulse generator
- function and control logic

Fig. 2-1: Block diagram of the SSI encoder



2.1.2 Applications

- Acquisition of displacement measuring systems
- X, Y, Z control
- Tolerance measurement

2.2 Signals used

The function "SSI" occupies **6 input and 2 ouput channels** of the function module on the board APCI-1710.

Table 2-1: Signals used

SIGNALS	ON CONNECTOR	POLARITY	FUNCTION
CLOCK_x	A**x** +/ -	Diff.	Clock output signal for the SSI encoders.
DATA1_x	B**x** +/ -	Diff. / OPT.24Vdiff.	DATA input for the first encoder
DATA2_x	C**x** + / -	Diff. / OPT. 24Vdiff.	DATA input for the second encoder
DATA3_x	D**x** + / -	Diff. / OPT.24Vdiff.	DATA input for the third encoder
Input1_x	E**x** +/-	24V / OPT. 5V	Digital input channel
Input2_x	F**x** +/-	24V / OPT. 5V	Digital input channel
Input3_x	G**x** +/-	24V / OPT. 5V	Digital input channel
Ouptut_x	H**x** +/-	24V / OPT. 5V TTL	Digital output channel

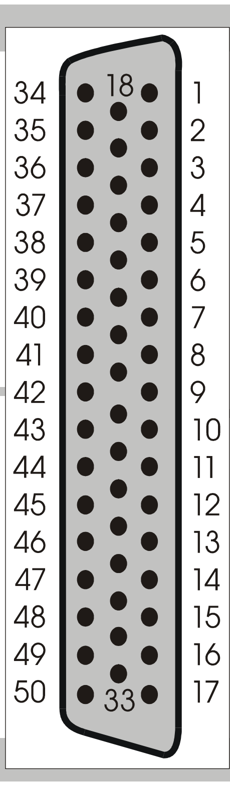
x : Number of the function module

2.3 Connector pin assignment

Fig. 2-2: Pin assignment of the 50-pin SUB-D male connector X1

	PIN	
Output	34	+ UREF
	35	H1
	36	H2
	37	H3
	38	H4
Input 1	39	E1
	40	E2
	41	E3
	42	E4
Input 2	43	F1
	44	F2
	45	F3
	46	F4
Input 3	47	G1
	48	G2
	49	G3
	50	G4

	PIN	
FUNCTION MODULE 3	18	A3 +
	19	A3 -
	20	B3 +
	21	B3 -
	22	C3 +
	23	C3 -
	24	D3 +
	25	D3 -
FUNCTION MODULE 4	26	A4 +
	27	A4 -
	28	B4 +
	29	B4 -
	30	C4 +
	31	C4 -
	32	D4 +
	33	D4 -



	PIN	
EXTGND	1	FUNCTION MODULE 0
A1 +	2	
A1 -	3	
B1 +	4	
B1 -	5	
C1 +	6	
C1 -	7	
D1 +	8	
D1 -	9	
A2 +	10	FUNCTION MODULE 1
A2 -	11	
B2 +	12	
B2 -	13	
C2 +	14	
C2 -	15	
D2 +	16	
D2 -	17	

2.4 Connections examples

The shift encoder CRE 58 by TWK is connected to the function module 1 of the APCI-/CPCI-1710and CPCI-1710. First interface.

Fig. 2-3: Connection to a shift encoder TWK CRE 58

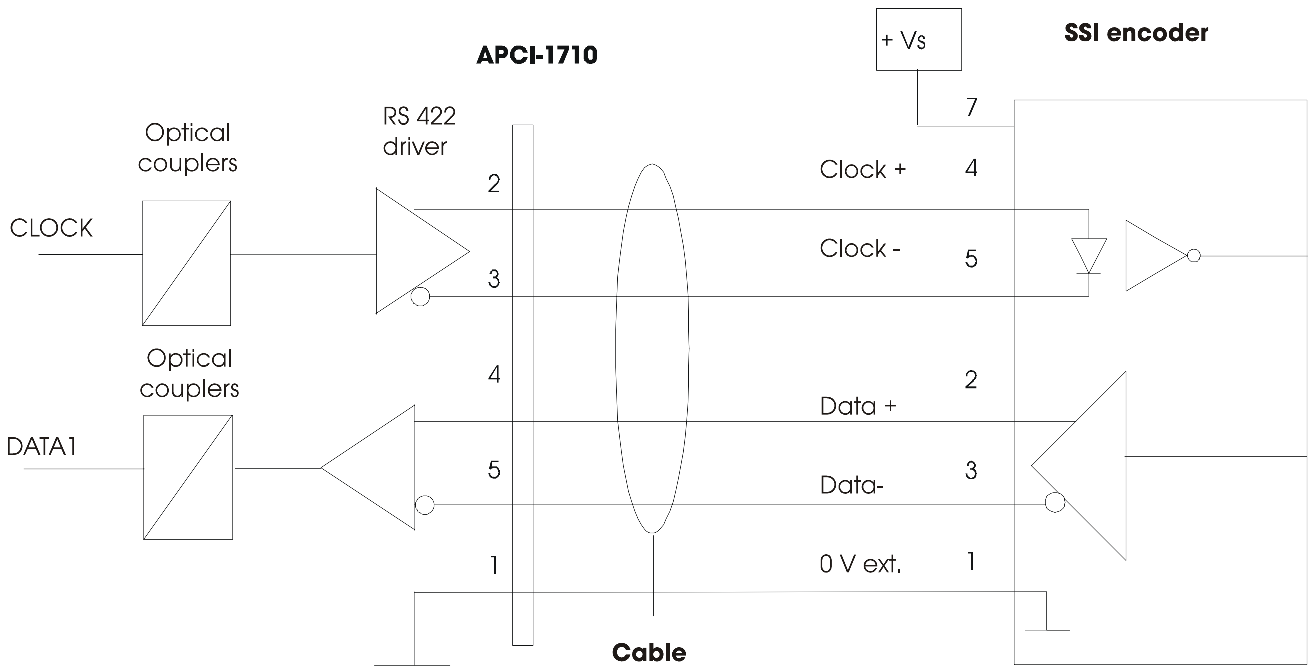
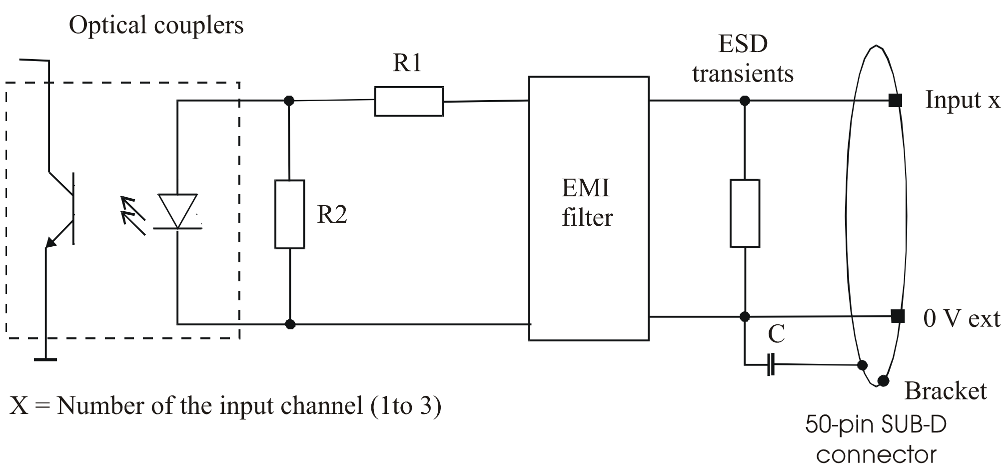
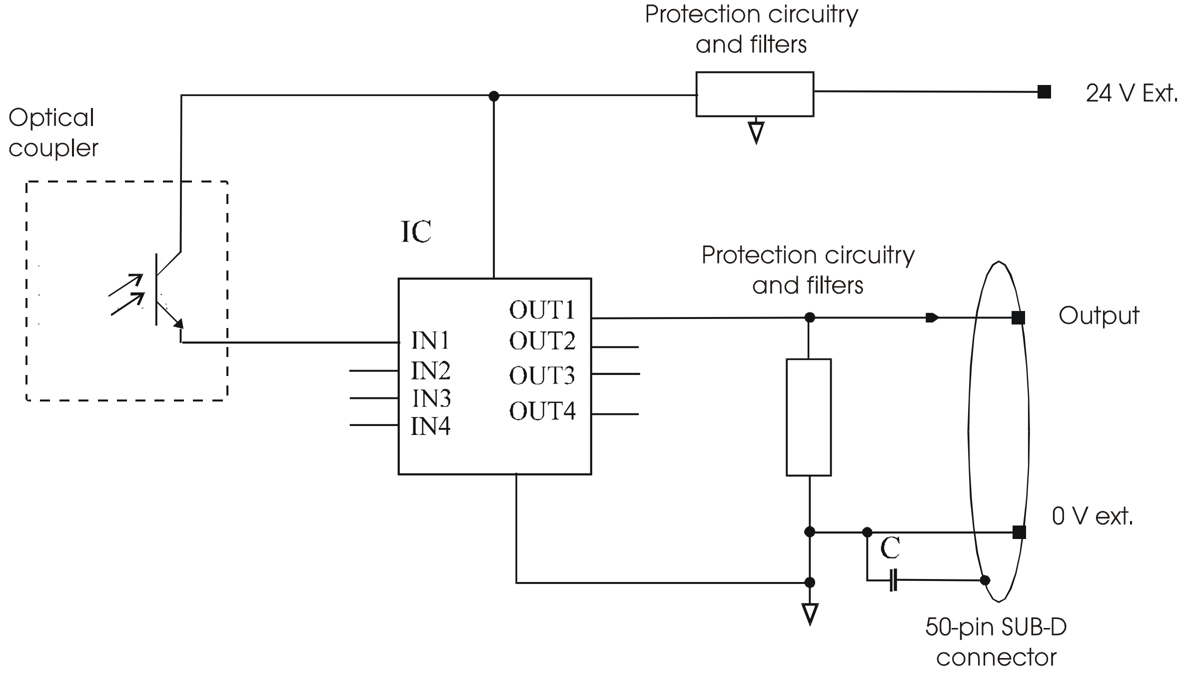


Fig. 2-4: Switching principle of the input channels



X = Number of the input channel (1to 3)

Fig. 2-5: Switching principle of the output channels



2.5 I/O mapping of the Synchronous Serial Interface

Table 2-2: I/O mapping of the Synchronous Serial Interface

	IORD			
	D31...D24	D23...D16	D15.....D8	D7.......D0
BYTES	HIGHBYTE	MIDHIGHBYTE	MIDLOWBYTE	LOWBYTE
BASE**x** + 0	-	-	-	STAUTS-REG
BASE**x** + 4	SHIFT1.3	SHIFT1.2	SHIFT1.1	SHIFT1.0
BASE**x** + 8	SHIFT2.3	SHIFT2.2	SHIFT2.1	SHIFT2.0
BASE**x** + 12	SHIFT3.3	SHIFT3.2	SHIFT3.1	SHIFT3.0
BASE**x** + 16	-	-	-	-
....	-	-	-	-
BASE**x** + 60	FUNKNBR2	FUNKNBR1	REVBYTE2	REVBYTE1

- : no function ; y : no significant data, **x :** number of the function module

The SSI occupies 5 DWORDS in the I/O range of the function module **x.**
The accesses are always read or written in 32-bit.

2.6 Description of the I/O functions

2.6.1 Function description

General description

The parallel and absolute information of the shift encoder are converted into serial information by an internal parallel-serial converter (Shift register).
They are then transmitted to the input channel in synchronisation with the clock emitted by the APCI-1710.
The synchronous transmission of the data word is introduced and controlled by a clock signal.
The length (i.e the variable) of the clock sequence is determined through an internal 8-bit register (COUNT-REG) in the APCI-/CPCI-1710so that the length of the data word to be transmitted can be changed from 0 to 32-bit.
For example, a shift encoder with a SSI 25-bit interface profile uses 26 clock signals to be read.
The clock frequency determines the velocity of the data transmission. This frequency is determined through an internal 16-bit-register.

Transmission protocol

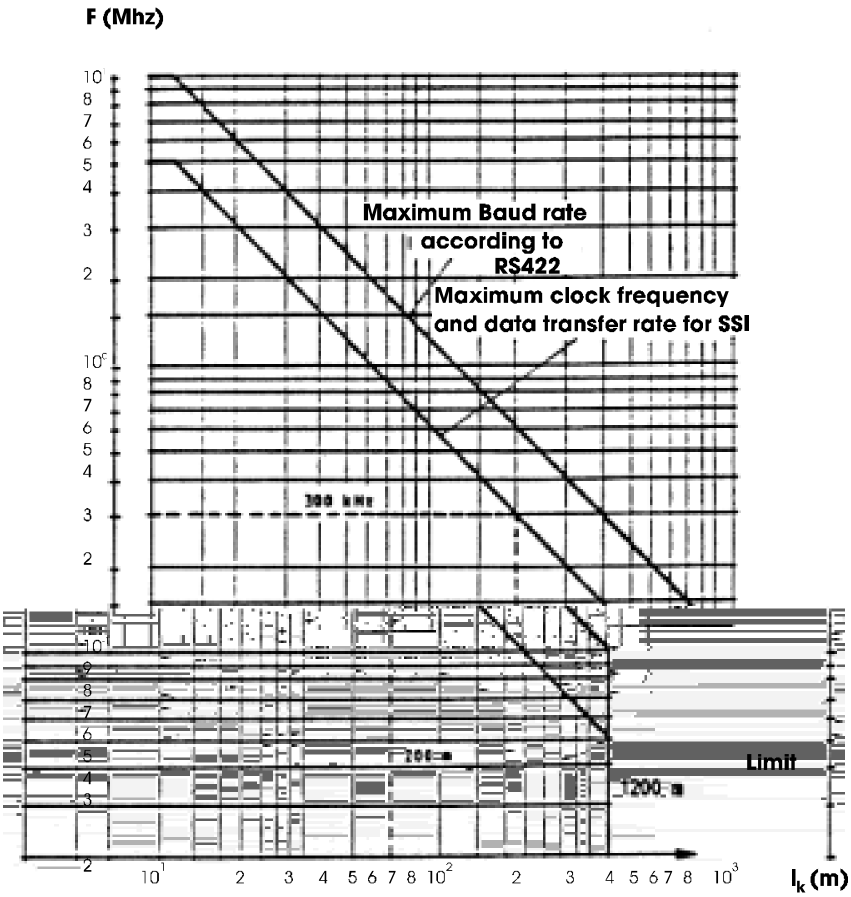
The logic levels refer to the TAKT+ (Clock) or DATA+ signals. In functionning or rest state the TAKT- (Clock) line and the data line (Clock+, DAT+) are Log1.
The input channel transmits the data transfer via a conversion of the clock signal from Log1 to Log0.
A retriggerable monoflop is set in the shift encoder through this change.
The output of this monoflop converts a paralllel shift register into a serial one and the parallel data which are present in gray-code are saved.

By the next change of the clock from Log0 to Log1, the most significant bit of the shift information is set on the data output channel of the shift encoder.
Any other rising edge drives the following bit to the output channel until the least significant bit. The clock of the monoflop is simultaneously retriggered with any falling edge.
The monoflop period (e.g. 20 µs) determines the period between 2 transfers and the minimum clock frequency.

Maximum data rates

Conditions:
The maximum data rate (clock frequency) is determined by the RS485 norm for the used drivers, receivers and transmission protocol.
The maximum clock frequency amounts half the values of the norm for the baud rate: i.e. 5 MHz

Fig. 2-6: Maximum data rate



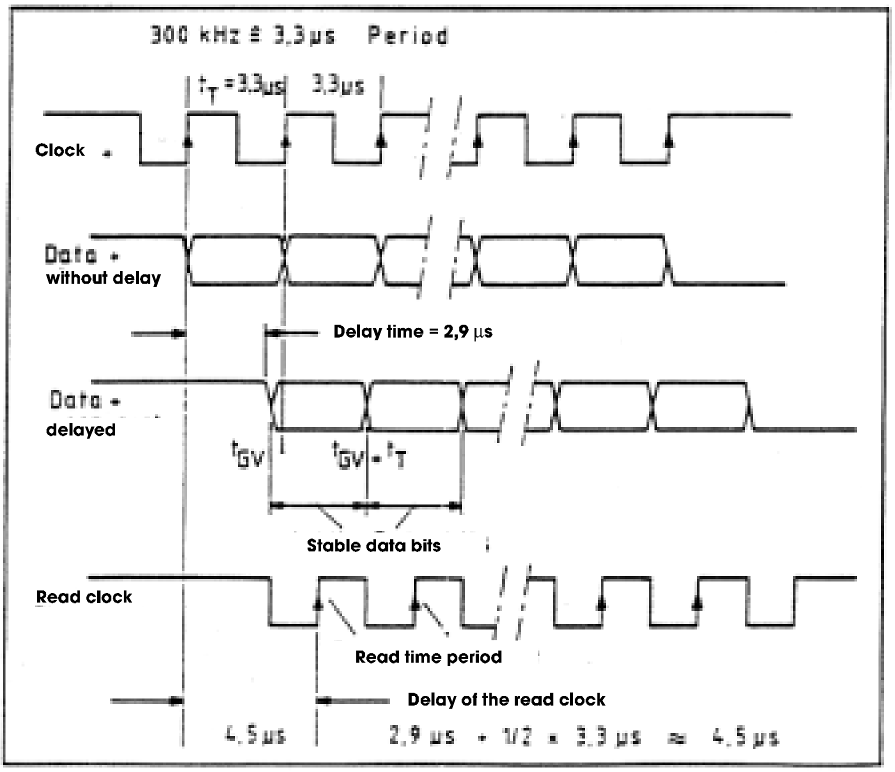
Transfer line:
The principal line of the SSI transfer consists in a shift encoder, a transmission cable and clock sequence and input channel. Each unit naturally transmits the signals with a delay time (running period). As a consequence the data present on the receiver is transmitted synchronously to the clock of the clock sequence yet with a delay amounting to the running period.
This delay time varies according to the length of the transmission line.
The clock rate is hence to be adapted to the line.

Delay times:
The total delay can be calculated as follows:

$$\boxed{Tgv = Te + 2 \times Tc + Tr}$$

Tgv: Total delay time =
Te: Delay time due to the electronics of the shift encoder, max. 150 ns
Tc: Cable delay: depends on the length of the cable. By using a cable of Type LIYCY-OB with 0.25 mm^2 cross section the specific running period amounts approx. to 6.5 ns/m.
Tr: Delay time of the receiver, max. 150 ns
Example:
For a cable length of 200 m the delay time amounts to:
Tdt (ns) = 300 + (2 x 6,5 x 200) = 2900 ns
The only selectable clock frequency is 300 kHz for a cable length of 200 m.

Fig. 2-7: Delay times



2.6.2 FRQ register (Base + 0)

On the base address +0 the 16-bit register determines the clock frequency.
This register can only be written.
Input frequency = PCI bus frequency (between 0 and 33 MHz. You will find the frequency in the manual of your PC).
If the register contains the value 0, the frequency is divided by 2.
If the register contains the value 50, the frequency is divided by 100.

These frequency values can be from 2 to 13070 in steps of 2.
Frequency values can be from 2 to 13070 in steps of 2.
Yet the frequency should not be higher than 1 MHz.

2.6.3 COUNTS register (Base + 4)

On the base address +4 the 8-bit register determines the number of bits for the SSI. This register can only be written.

Standard SSI 25-bit use 26 clock signals
If the register contains the value n = 25, 26 clock signals are generated;
If the register contains the value n = 32, 33 clock signals are generated.

Transmission example of a 18-bit shift encoder

Encoder with 1024 steps / turn (10-bit in monotour) and 256 turns (8-bit in multitour).
The transmission protocol is delivered in the standard execution for a 25-bit data word. 12 bits are for the number of turns, 13 for the resolution (step/turn).
As the transmission always begins with the multitour bit 12 and the multitour is determined for 8-bit in this example, 4 spaces are transmitted fisrt with Lo0, then 8 bits of the multitour.
The bits of the monotour (S10 to S1) are following. The empty 3 bit are also transmitted with Log0.

Table 2-3: Transmission for a 18-bit shift encoder

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25		
4096	12	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	8192	13
2048	11	0	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	0	4096	12
1024	10	0	0	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	0	0	2048	11
512	9	0	0	0	M9	M8	M7	M6	M5	M4	M3	M2	M1	**S10**	**S9**	**S8**	**S7**	**S6**	**S5**	**S4**	**S3**	**S2**	**S1**	**0**	**0**	**0**	**1024**	**10**
256	**8**	**0**	**0**	**0**	**0**	**M8**	**M7**	**M6**	**M5**	**M4**	**M3**	**M2**	**M1**	S9	S8	S7	S6	S5	S4	S3	S2	S1	0	0	0	0	512	9
128	7	0	0	0	0	0	M7	M6	M5	M4	M3	M2	M1	S8	S7	S6	S5	S4	S3	S2	S1	0	0	0	0	0	256	8
64	6	0	0	0	0	0	0	M6	M5	M4	M3	M2	M1	S7	S6	S5	S4	S3	S2	S1	0	0	0	0	0	0	128	7
32	5	0	0	0	0	0	0	0	M5	M4	M3	M2	M1	S6	S5	S4	S3	S2	S1	0	0	0	0	0	0	0	64	6
16	4	0	0	0	0	0	0	0	0	M4	M3	M2	M1	S5	S4	S3	S2	S1	0	0	0	0	0	0	0	0	32	5
8	3	0	0	0	0	0	0	0	0	0	M3	M2	M1	S4	S3	S2	S1	0	0	0	0	0	0	0	0	0	16	4
4	2	0	0	0	0	0	0	0	0	0	0	M2	M1	S3	S2	S1	0	0	0	0	0	0	0	0	0	0	8	3
2	1	0	0	0	0	0	0	0	0	0	0	0	M1	S2	S1	0	0	0	0	0	0	0	0	0	0	0	4	2
Number of turns	Bit/turn	Multitour bit												Monotour bit													Steps/turn	Bit/turn

2.6.4 CONTROL register (Base + 12)

On the base address +12 a 8-bit register allows to convert from GRAY code into BINARY code.
This register can only be written.
The release occurs through a bit for each SSI module. If the 3 bits are set to "0" after reset no data conversion is completed.
If one of the three bits is set to „1“, in the corresponding SSI module the GRAY code is converted to the BINARY code.

2.6.5 START register (Base +8)

By writing on the base address +8 a cycle begins. The data are serially transmitted.

2.6.6 STATUS register (Base + 0)

The information about the current serial data transfer is read on the base address +0.
The state of the digital input channels is accessed by a read command.

Table 2-4: Status register

BIT D0	0	Tranfer is over; data is ready in the SHIFT-REGISTER
	1	Transfer
BIT D1	0	SSI data input is on low level
	1	SSI data input is on high level
BIT D2	0	SSI data input is on low level
	1	SSI data input is on high level
BIT D3	0	SSI data input is on low level
	1	SSI data input is on high level
BIT D4	0	Input1 is on high level
	1	Input1 is on low level
BIT D5	0	Input2 is on high level
	1	Input2 is on low level
BIT D6	0	Input3 is on high level
	1	Input3 is on low level

2.6.7 SHIFT register

3 x 32-bit registers are available to save the data of the SSI.
The reading operation occurs in 32-bit.
The data from the 32-bit registers is then filtered (See table 2-1).

2.6.8 OUTPUT register

When the bit D0 is set, the output channel is set.
When the bit D0 is reset (State after reset), the output channel is disabled.

2.6.9 IDENTITY register (Base + 60)

On the base address +60 a reading operation indicates the function as well as the revision of the function in ASCI format.

BASE + 60	"S"	"I"	"1"	"0"

Means: Synchronous Serial Interface Rev 1.0

2.7 Operating with a Synchronous Serial Interface

The transfer profile of the SSI encoder determines the number of necessary pulses for the transfer of the position information
For example: 25-bit profile uses 26 clocks → in the COUNTS register 25 clock signals are to be written.
Clock frequency:
- the minimum frequency is determined by the monoflop time. For approx. 20 µS the minimum frequency is 50 kHz
- the maximum frequency is determined by the length of the cable used. The divider factor is written in the FRQ register.

1. The new values of the SSI encoder are demanded through a dummy writing on the START register.
2. The end of the transmission is demanded through the DONE bit in the STATUS register (Polling).
3. read the values in the SHIFT register.

3 DEVICE DRIVER

3.1 Introduction

i

IMPORTANT!
Please note the following comventions in the text:

Function: "i_APCI1710_SetBoardInformation"
Variable *ui_Address*

Table 3-1: Define value

Define name	Decimal value	Hexadecimal value
DLL_COMPILER_C	1	1
DLL_COMPILER_VB	2	2
DLL_COMPILER_PASCAL	3	3
DLL_LABVIEW	4	4
APCI1710_DISABLE	0	0
APCI1710_ENABLE	1	1
ACPI1710_30MHZ	30	1E
APCI1710_33MHZ	33	21
APCI1710_BINARY_MODE	1	1
APCI1710_GRAY_MODE	0	0

3.2 SSI initialisation

1) i_APCI1710_InitSSI (...)

Syntax:

```
<Return value> = i_APCI1710_InitSSI
                  (BYTE    b_BoardHandle,
                   BYTE    b_ModulNbr,
                   BYTE    b_SSIProfile,
                   BYTE    b_PositionTurnLength,
                   BYTE    b_TurnCptLength,
                   BYTE    b_PCIInputClock,
                   ULONG   ul_SSIOutputClock,
                   BYTE    b_SSICountingMode)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **APCI-/CPCI-1710** board
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_SSIProfile	Selection of the SSI profile length (2 to 32). See Fig. 4-1.
BYTE	b_PositionTurnLength	Selection of the SSI position data length. (1 to 31) See Fig. 4-1.
BYTE	b_TurnCptLength	Selection of the SSI counter data length (1 to 31). See Fig. 4-1.
BYTE	b_PCIInputClock	Selection of the PCI bus clock - APCI1710_30MHZ : The PC has a PCI bus clock of 30 MHz - APCI1710_33MHZ : The PC has a PCI bus clock of 33 MHz
ULONG	ul_SSIOutputClock	Selection of the SSI output clock. From 229 to 5 000 000 Hz for 30 MHz selection. From 252 to 5 000 000 Hz for 33 MHz selection.
BYTE	b_SSICountingMode	SSI counter mode selection. - APCI1710_BINARY_MODE : Binary counter mode. - APCI1710_GRAY_MODE : Gray counter mode.

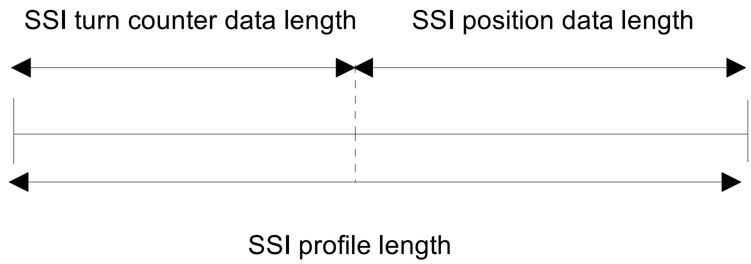
- Output:

No output signal has occurred

Task:

Configures the SSI operating mode of the selected module (*b_ModulNbr*). You must call up this function before you call up any other function to access the SSI.

Fig. 3-1: SSI profile



Calling convention:

ANSI C :

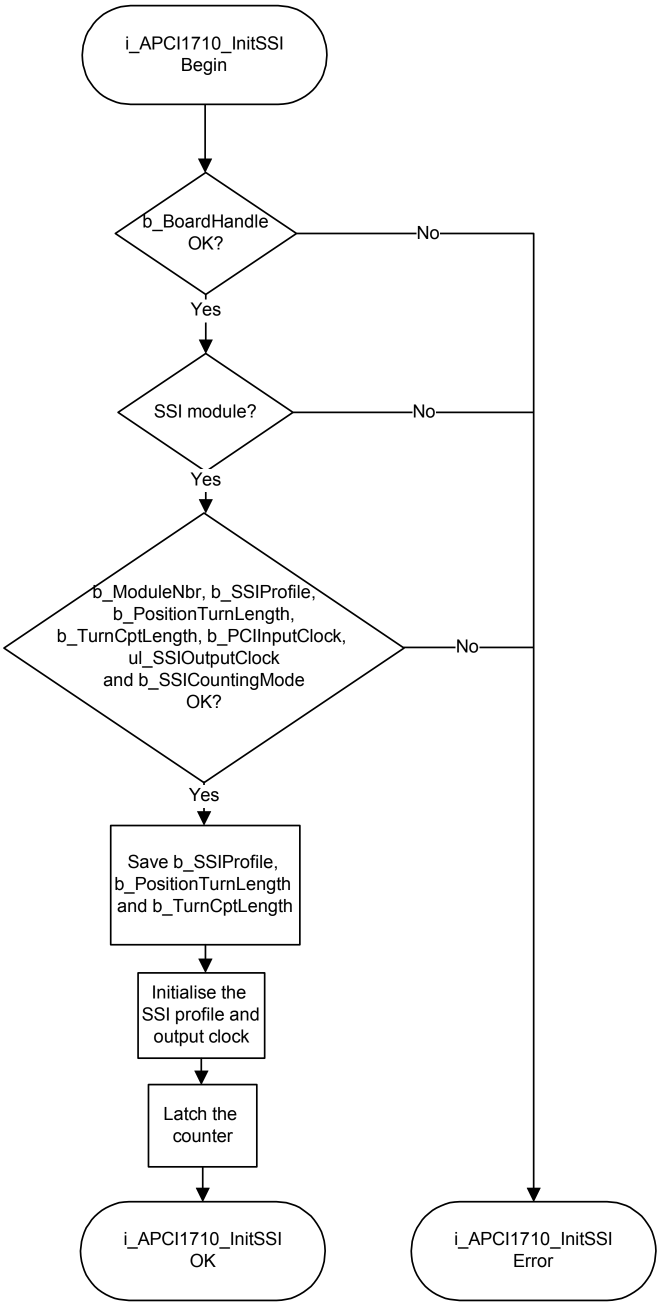
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_APCI1710_InitSSI(b_BoardHandle,
                                   0,
                                   25,
                                   12,
                                   12,
                                   APCI1710_33MHZ,
                                   150000,
                                   APCI1710_BINARY_MODE);
```

Return value:

0: No error.
-1: The handle parameter of the board is wrong.
-2: Module selection is wrong.
-3: The module is not a SSI module.
-4: The selected SSI profile length is wrong.
-5: The selected SSI position data length is wrong.
-6: The selected SSI turn counter data length is wrong.
-7: The selected PCI input clock is wrong.
-8: The selected SSI output clock is wrong.
-9: The selected SSI counter mode parameter is wrong.

Input	
b_BoardHandle	b_TurnCptLength
b_ModulNbr	b_PCIInputClock
b_SSIProfil	ul_SSIOutputClock
b_PositionTurnLength	b_SSICountingMode

Function diagram



Output

<Return Value>

3.3 Read SSI

1) i_APCI1710_Read1SSIValue (...)

Syntax:

```
<Return value> = i_APCI1710_Read1SSIValue
                        (BYTE     b_BoardHandle,
                         BYTE     b_ModulNbr,
                         BYTE     b_SelectedSSI,
                         PULONG   pul_Position,
                         PULONG   pul_TurnCpt)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **APCI-/CPCI-1710** board
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_SelectedSSI	Selection of the SSI counter (0 to 2)

- Output:

PULONG	pul_Position	SSI position in the turn
PULONG	pul_TurnCpt	Number of turns

Task:

Reads the SSI counter (*b_SelectedSSI*) of the selected module (*b_ModulNbr*).

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned long   ul_Position;
unsigned long   ul_TurnCpt;

i_ReturnValue = i_APCI1710_Read1SSIValue     (b_BoardHandle,
                                               0,
                                               0,
                                               &ul_Position,
                                               &ul_TurnCpt);
```

Return value:

0: No error.

-1: The handle parameter of the board is wrong.

-2: Module selection is wrong.

-3: The module is not a SSI module.

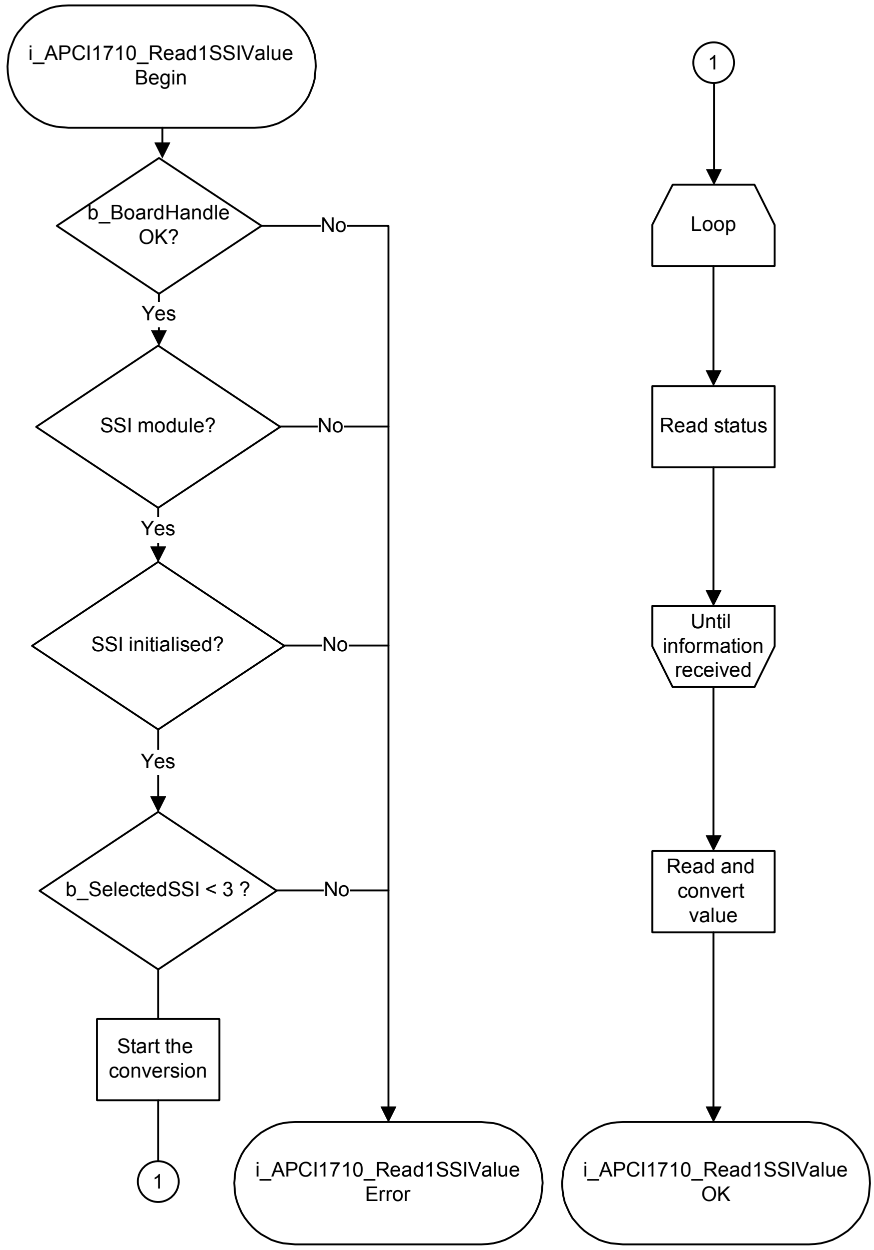
-4: SSI not initialised see function " i_APCI1710_InitSSI".

-5: The selected SSI is wrong.

Input

b_BoardHandle	pul_Position
b_ModulNbr	pul_TurnCpt
b_SelectedSSI	

Function diagram



Output

pul_Position
pul_TurnCpt

<Return Value>

2) i_APCI1710_ReadAllSSIValue (...)

Syntax:

```
<Return value> = i_APCI1710_ReadAllSSIValue
                        (BYTE     b_BoardHandle,
                         BYTE     b_ModulNbr,
                         PULONG   pul_Position,
                         PULONG   pul_TurnCpt)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **APCI-/CPCI-1710** board
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

- Output:

PULONG	pul_Position	SSI position array in the turn
PULONG	pul_TurnCpt	Number of turns

Task:

Read all SSI counters of the selected module (*b_ModulNbr*).
pul_Position [0] : SSI position of the SSI counter 0
pul_Position [1] : SSI position of the SSI counter 1
pul_Position [2] : SSI position of the SSI counter 2
pul_TurnCpt [0] : Number of countings of the SSI counter 0
pul_TurnCpt [1] : Number of countings of the SSI counter 1
pul_TurnCpt [2] : Number of countings of the SSI counter 2

Calling convention:

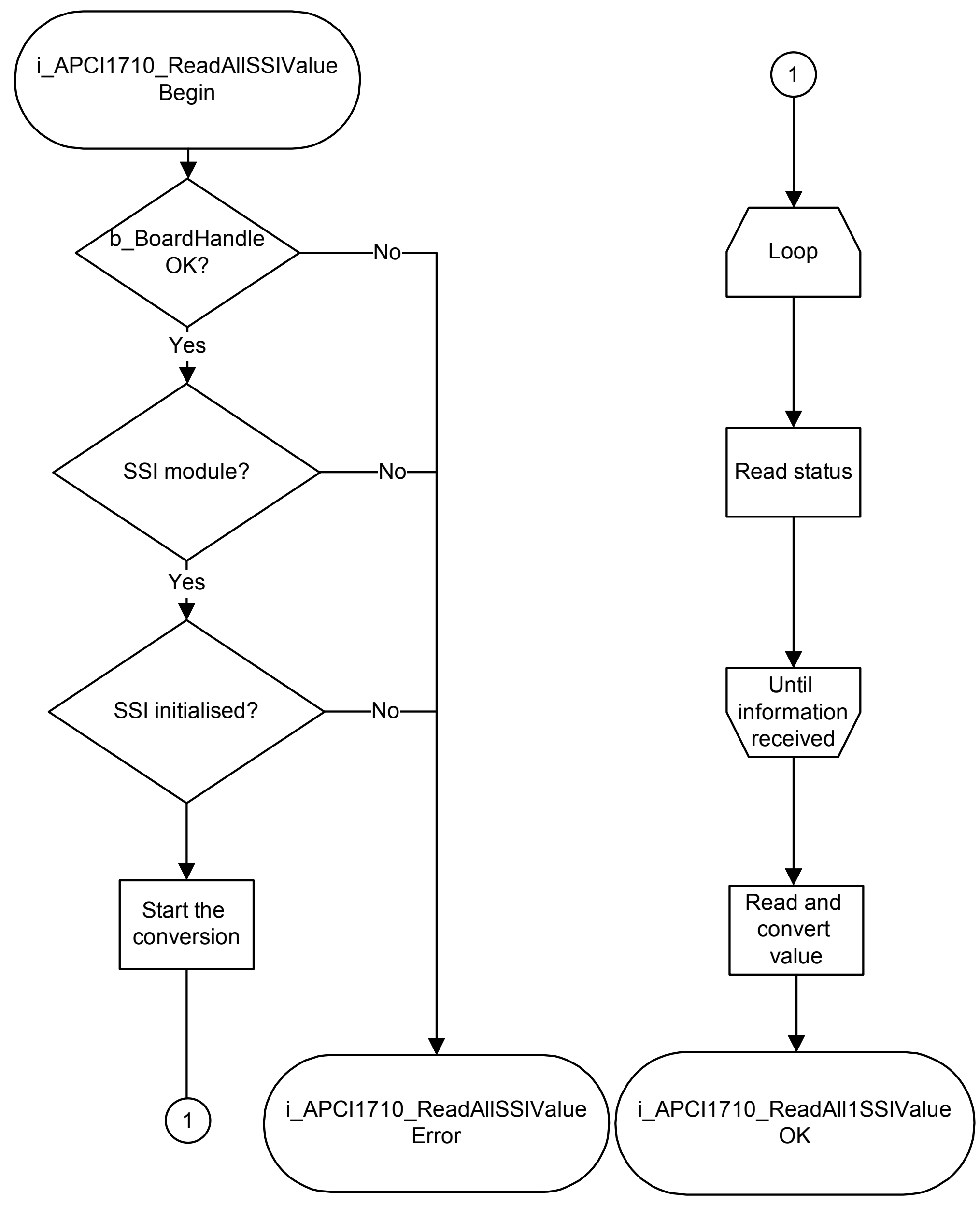
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned long   ul_Position   [3];
unsigned long   ul_TurnCpt  [3];

i_ReturnValue = i_APCI1710_ReadAllSSIValue    (b_BoardHandle,
                                               0,
                                               ul_Position,
                                               ul_TurnCpt);
```

Return value:

0: No error.
-1: The handle parameter of the board is wrong.
-2: Module selection is wrong.
-2: The module is not a SSI module.
-3: SSI not initialised see function " i_APCI1710_InitSSI".

Input	
b_BoardHandle	b_ModulNbr
Function diagram	



Output
pul_Position pul_TurnCpt
<Return Value>

3.4 Digital SSI inputs

1) i_APCI1710_ReadSSI1DigitalInput (...)

Syntax:

```
<Return value> = i_APCI1710_ReadSSI1DigitalInput
                        (BYTE      b_BoardHandle,
                         BYTE      b_ModulNbr,
                         BYTE      b_InputChannel,
                         PBYTE     pb_ChannelStatus)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **APCI-/CPCI-1710**
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_InputChannel	Selection of the digital input (0 to 2).

- Output:

PBYTE	pb_ChannelStatus	Digital input channel status. 0 : channel is not active 1 : channel is active

Task:

Returns the status of the selected digital input (*b_InputChannel*) to the SSI module (*b_ModulNbr*).

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_ChannelStatus;

i_ReturnValue = i_APCI1710_ReadSSI1DigitalInput    (b_BoardHandle,
                                                     0,
                                                     0,
                                                    &b_ChannelStatus);
```

Return value:

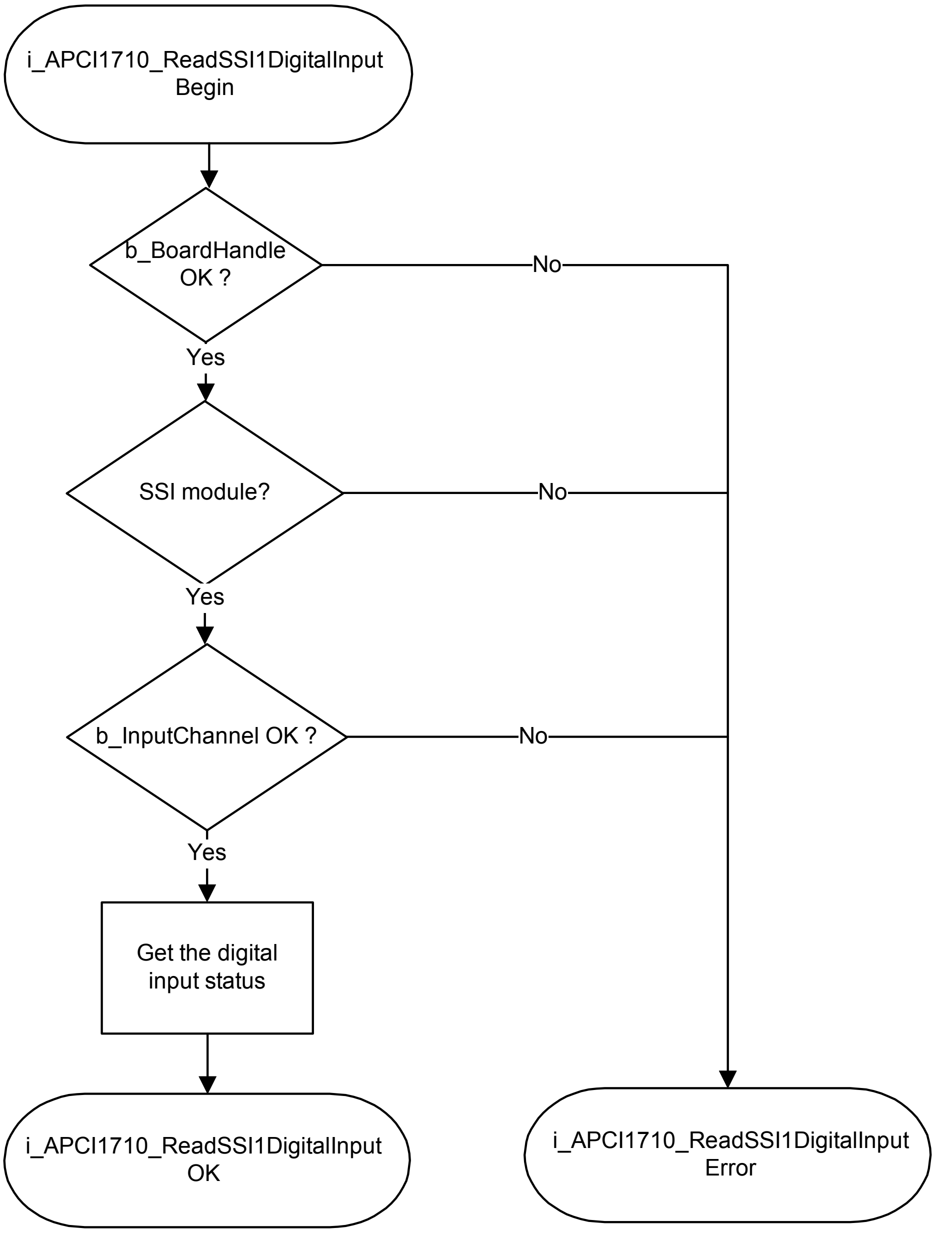
0: No error.

-1: The handle parameter of the board is wrong.

-2: Module selection is wrong.

-3: The module is not a SSI module.

-4: The selected SSI digital input selection is wrong.

Input	
b_BoardHandle b_ModulNbr	b_InputChannel
Function diagram	
	
Output	
pb_ChannelStatus	
<Return Value>	

2) i_APCI1710_ReadSSIAllDigitalInput (...)

Syntax:

```
<Return value> = i_APCI1710_ReadSSIAllDigitalInput
                        (BYTE       b_BoardHandle,
                         BYTE       b_ModulNbr,
                         PBYTE      pb_InputStatus)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **APCI-/CPCI-1710**
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

- Output:

PBYTE	pb_InputStatus	Status of the digital input channels.

Task:

Returns the status of all digital inputs from selected SSI module (*b_ModulNbr*).

D2	D1	D0
INPUT 2	INPUT 1	INPUT 0

0 : channel is not active
1 : channel is active

Calling convention:

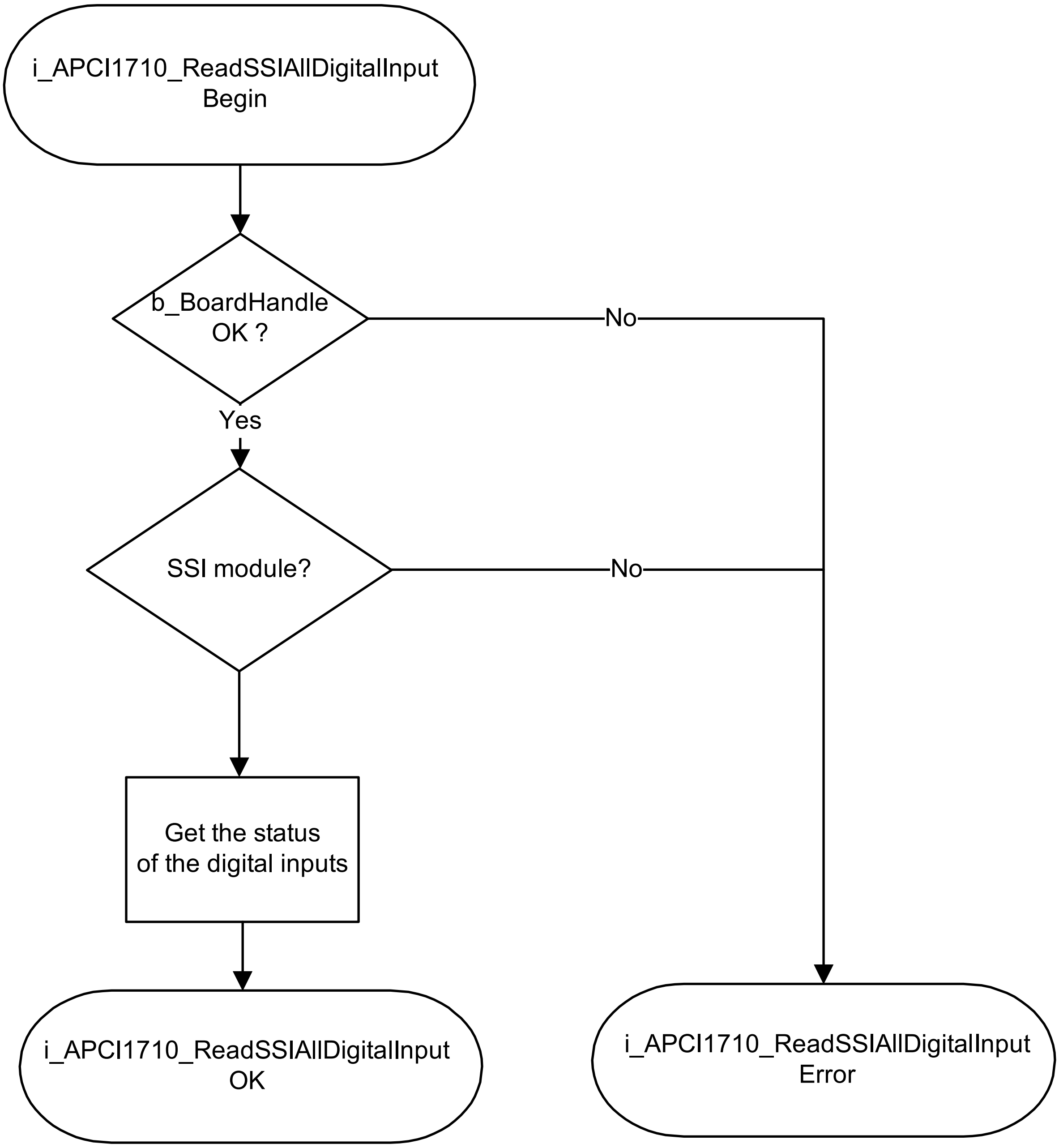
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_InputStatus;

i_ReturnValue = i_APCI1710_ReadSSIAllDigitalInput (b_BoardHandle,
                                                   0,
                                                   &b_InputStatus);
```

Return value:

0: No error.
-1: The handle parameter of the board is wrong.
-2: Module selection is wrong.
-3: The module is not a SSI module.

Input
b_BoardHandle b_ModulNbr
Function diagram



Output
pb_InputStatus
<Return Value>

3.5 Digital SSI outputs

1) i_APCI1710_SetSSIDigitalOutputOn (...)

Syntax:

```
<Return value> = i_APCI1710_SetSSIDigitalOutputOn
                                        (BYTE      b_BoardHandle,
                                         BYTE      b_ModulNbr)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **APCI-/CPCI-1710**
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

- Output:

No output signal has occurred

Task:

Sets the digital output of the selected SSI module (b_ModulNbr) ON.

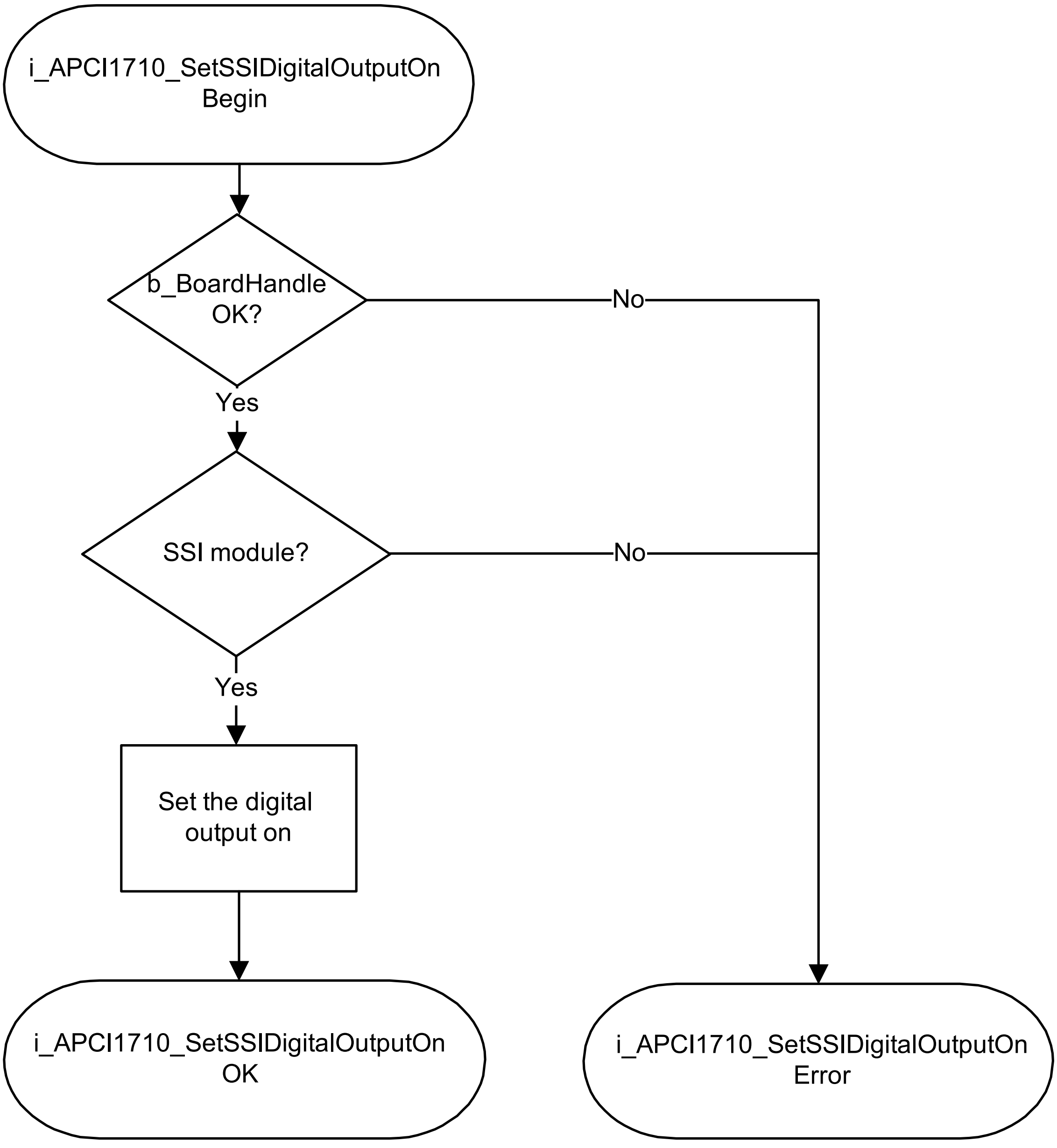
Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_APCI1710_SetSSIDigitalOutputOn   (b_BoardHandle,
                                                     0);
```

Return value:

0: No error.
-1: The handle parameter of the board is wrong.
-2: Module selection is wrong.
-3: The module is not a SSI module.

Input
b_BoardHandle b_ModulNbr
Function diagram

Output
<Return Value>

2) i_APCI1710_SetSSIDigitalOutputOff (...)

Syntax:

<Return value> = i_APCI1710_SetSSIDigitalOutputOff
(BYTE b_BoardHandle,
BYTE b_ModulNbr)

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **APCI-/CPCI-1710**
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

- Output:

No output signal has occurred.

Task:

Sets the digital output of the selected SSI module (b_ModulNbr) OFF.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_APCI1710_SetSSIDigitalOutputOff (b_BoardHandle,
                                                   0);
```

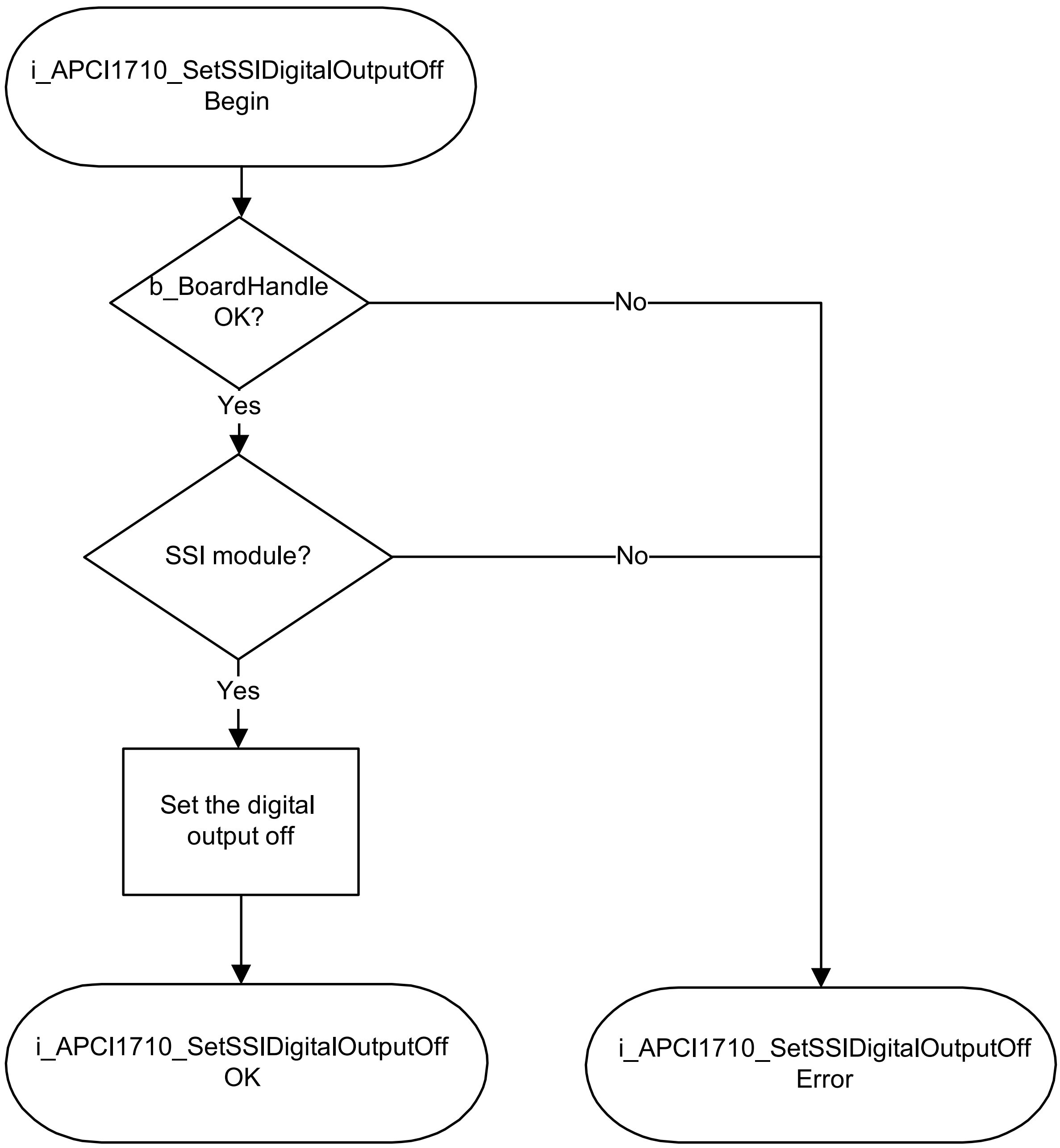
Return value:

0: No error.
-1: The handle parameter of the board is wrong.
-2: Module selection is wrong.
-3: The module is not a SSI module.

Input

b_BoardHandle
b_ModulNbr

Function diagram



Output

<Return Value>

3.6 Kernel functions

IMPORTANT!
These functions are only available for the Windows NT and Windows 95 user interrupt routine in the synchronous mode.
See function "i_APCI1710_SetBoardIntRoutineWin32"

1) i_APCI1710_KRNL_ReadSSI1DigitalInput (...)

Syntax:

```
<Return value> = i_APCI1710_KRNL_ReadSSI1DigitalInput
                        (UINT     ui_BaseAddress,
                         BYTE     b_ModulNbr,
                         BYTE     b_InputChannel,
                         PBYTE    pb_ChannelStatus)
```

Parameters:

- Input:

UINT	ui_BaseAddress	**APCI-/CPCI-1710**base address See " i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_InputChannel	Selection of the digital input channel (0 to 2).

- Output:

PBYTE	pb_ChannelStatus	Digital input channel status. 0 : channel is not active 1 : channel is active

Task:

Returns the status of the selected digital input (*b_InputChannel*) to the SSI module (*b_ModulNbr*).

Calling convention:

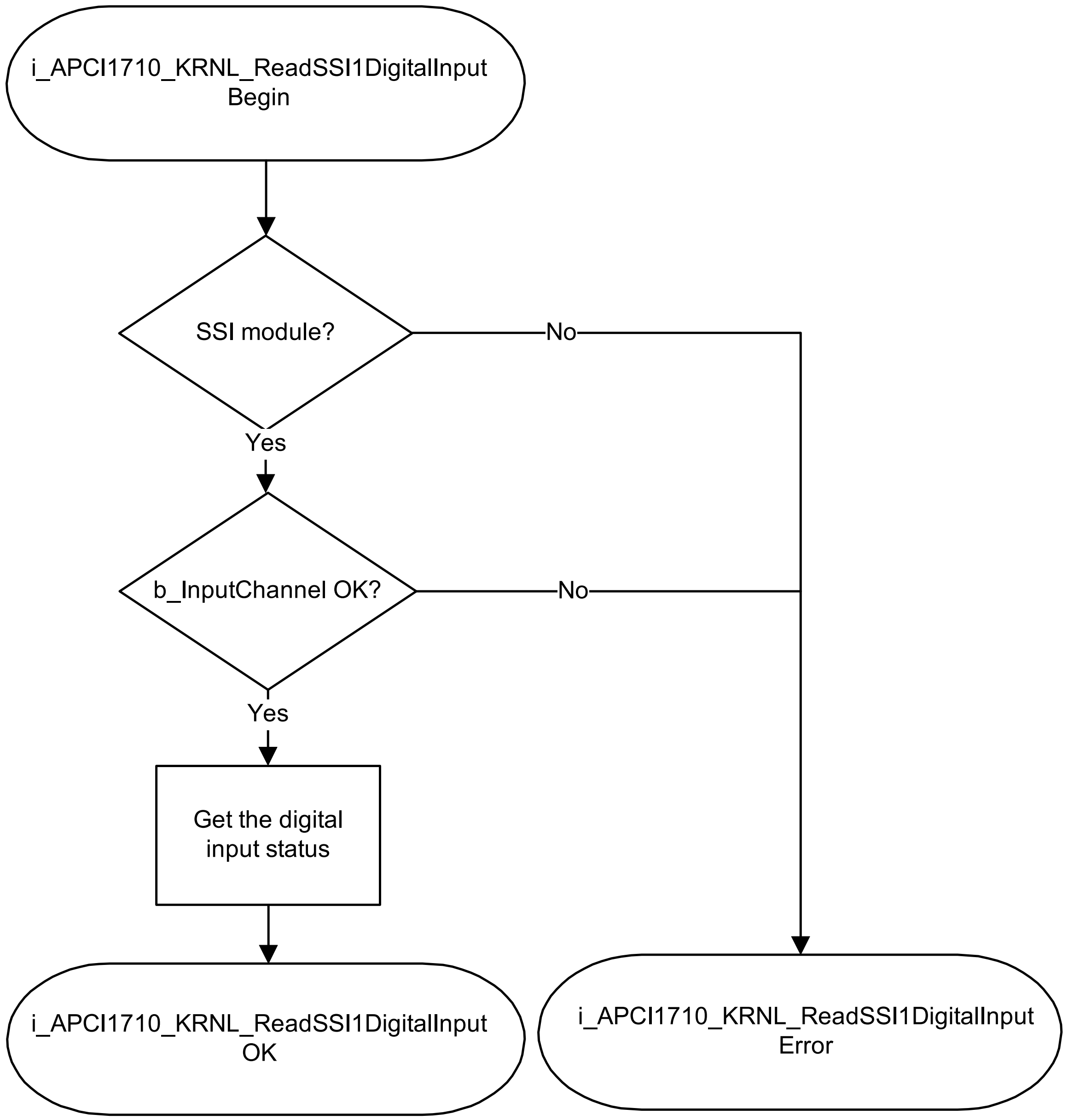
ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;
unsigned char   b_ChannelStatus;

i_ReturnValue = i_APCI1710_KRNL_ReadSSI1DigitalInput
                                        (ui_BaseAddress,
                                         0,
                                         0,
                                         &b_ChannelStatus);
```

Return value:

0: No error.
-1: Module selection is wrong.
-2: The module is not a SSI module.
-3: The selected SSI digital input is wrong.

Input	
ui_BaseAddress b_ModulNbr	b_InputChannel
Function diagram	



Output
pb_ChannelStatus
<Return Value>

2) i_APCI1710_KRNL_ReadSSIAllDigitalInput (...)

Syntax:

```
<Return value> = i_APCI1710_KRNL_ReadSSIAllDigitalInput
                         (UINT       ui_BaseAddress,
                          BYTE       b_ModulNbr,
                          PBYTE      pb_InputStatus)
```

Parameters:

- Input:

UINT	ui_BaseAddress	**APCI-/CPCI-1710**base address See " i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

- Output:

PBYTE	pb_InputStatus	Status of the digital inputs channels

Task:

Returns the status of all digital inputs to the selected SSI module (*b_ModulNbr*).

D2	D1	D0
INPUT 2	INPUT 1	INPUT 0

0 : Channel is not active
1 : Channel is active

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;
unsigned char   b_InputStatus;

i_ReturnValue = i_APCI1710_KRNL_ReadSSIAllDigitalInput
                                  (ui_BaseAddress,
                                   0,
                                   &b_InputStatus);
```

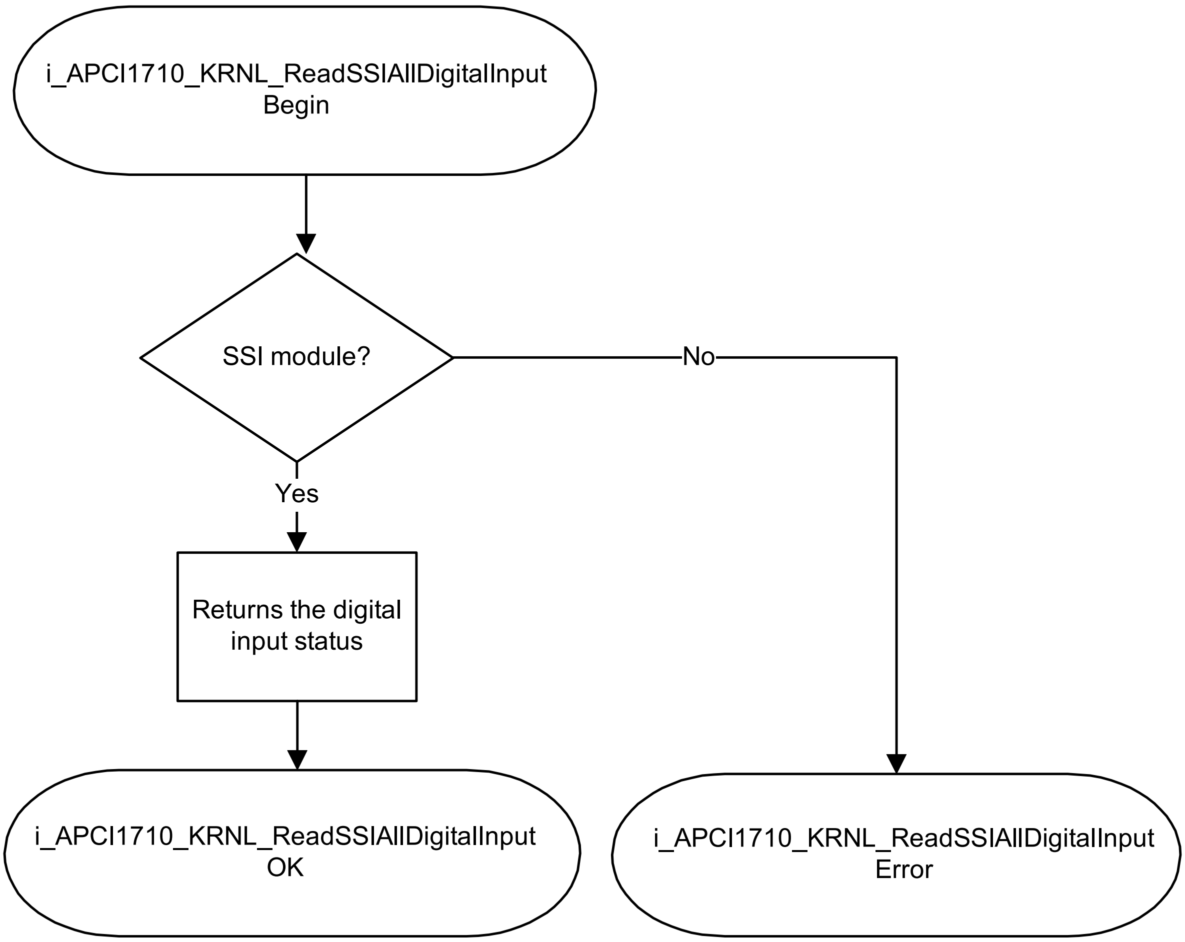
Return value:

0: No error.
-1: Module selection is wrong.
-2: The module is not a SSI module.

Input

ui_BaseAddress
b_ModulNbr

Function diagram



Output

pb_InputStatus

<Return Value>

3) i_APCI1710_KRNL_SetSSIDigitalOutputOn (...)

Syntax:

```
<Return value> = i_APCI1710_KRNL_SetSSIDigitalOutputOn
                                  (UINT        ui_BaseAddress,
                                   BYTE        b_ModulNbr)
```

Parameters:

- Input:

UINT	ui_BaseAddress	**APCI-/CPCI-1710**base address See " i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

- Output:

No output signal has occurred.

Task:

Sets the digital output of the selected SSI module (b_ModulNbr) on.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;

i_ReturnValue = i_APCI1710_KRNL_SetSSIDigitalOutputOn
                                          (ui_BaseAddress,
                                           0);
```

Return value:

0: No error.

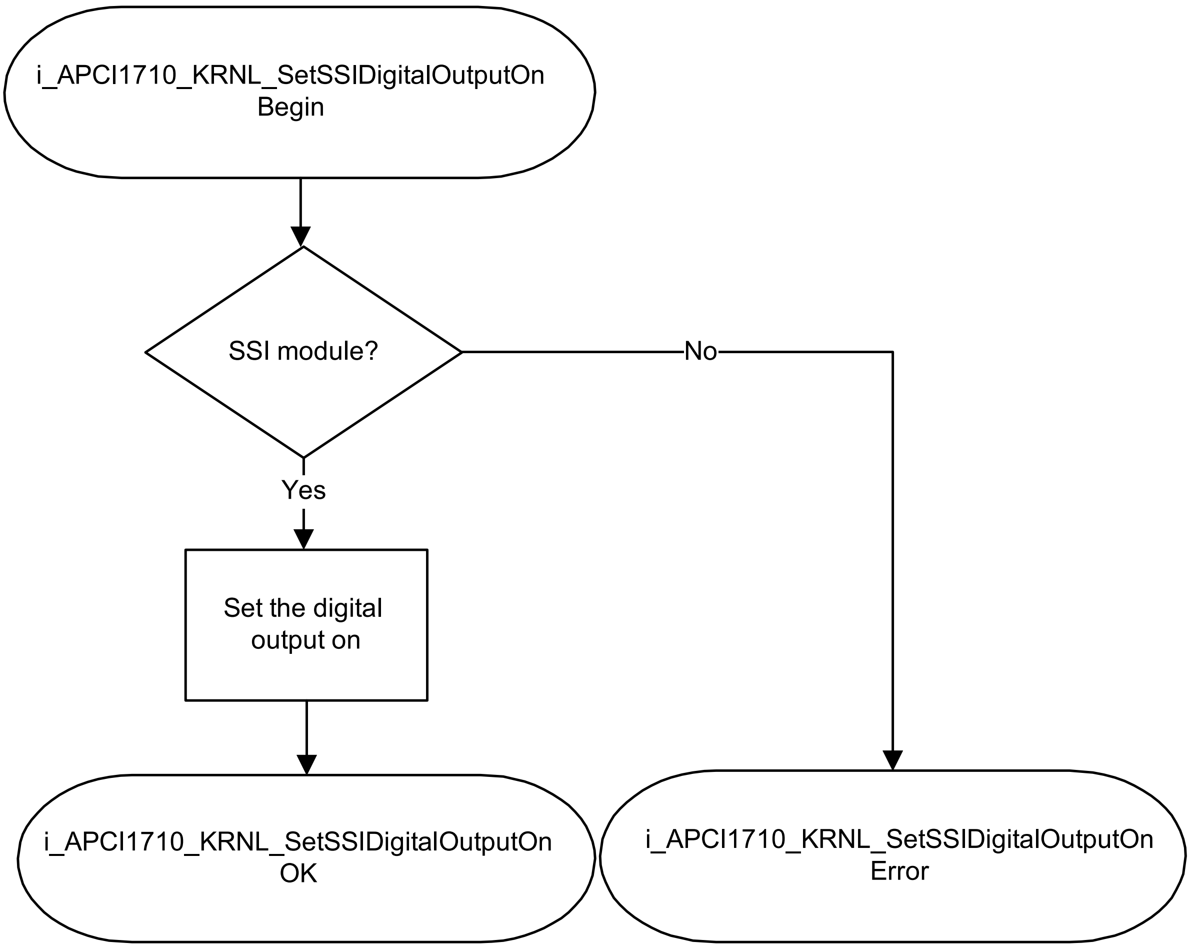
-1: Module selection is wrong.

-2: The module is not a SSI module.

Input

ui_BaseAdress
b_ModulNbr

Function diagram



Output

<Return Value>

4) i_APCI1710_KRNL_SetSSIDigitalOutputOff (...)

Syntax:

```
<Return value> = i_APCI1710_KRNL_SetSSIDigitalOutputOff
                              (UINT       ui_BaseAddress,
                               BYTE       b_ModulNbr)
```

Parameters:

- Input:

UINT	ui_BaseAddress	**APCI-/CPCI-1710**base address See “ i_APCI1710_GetHardwareInformation“
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

- Output:

No output signal has occurred.

Task:

Sets the digital output of the selected SSI module (b_ModulNbr) off.

Calling convention:

ANSI C :

```
int            i_ReturnValue;
unsigned int   ui_BaseAddress;

i_ReturnValue = i_APCI1710_KRNL_SetSSIDigitalOutputOff
                                        (ui_BaseAddress,
                                         0);
```

Return value:

0: No error.

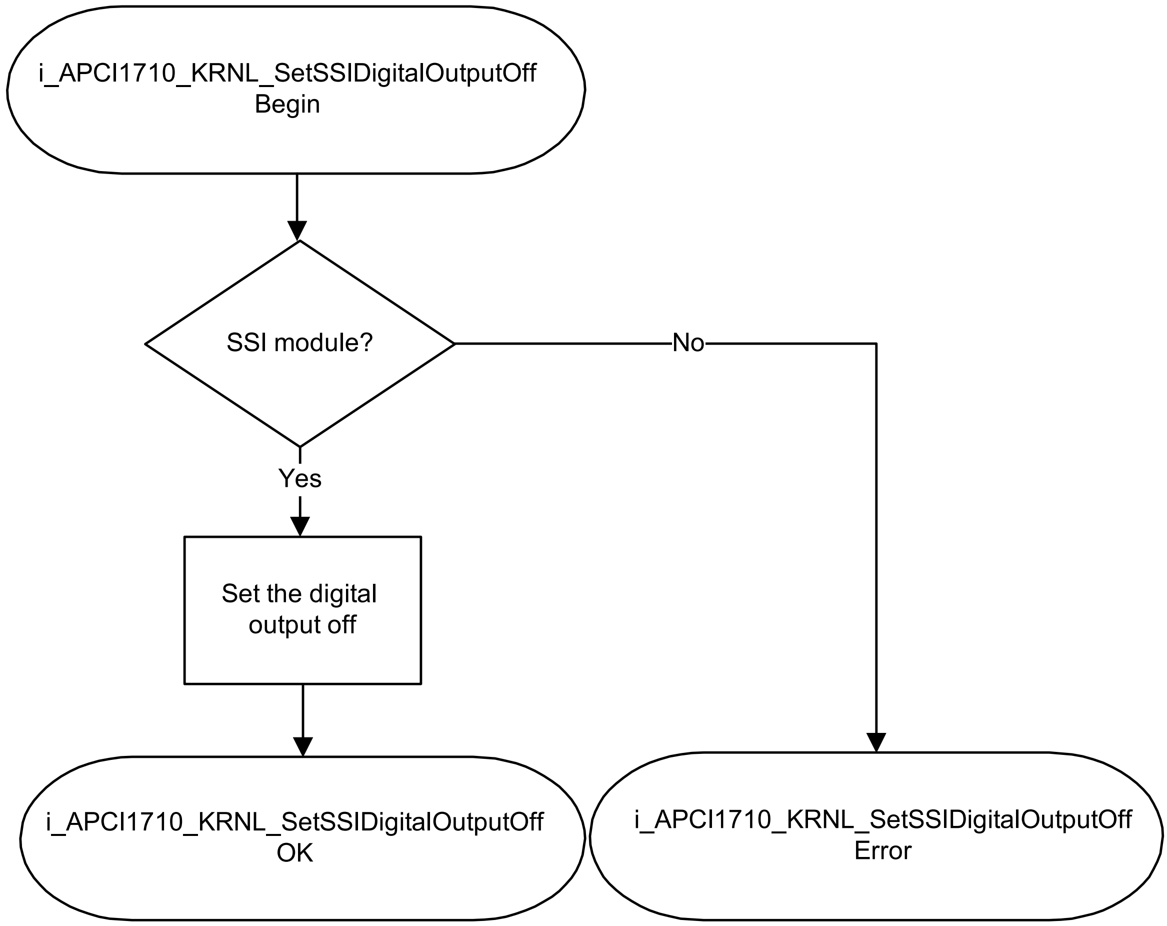
-1: Module selection is wrong.

-2: The module is not a SSI module.

Input

ui_BaseAddress
b_ModulNbr

Function diagram



Output

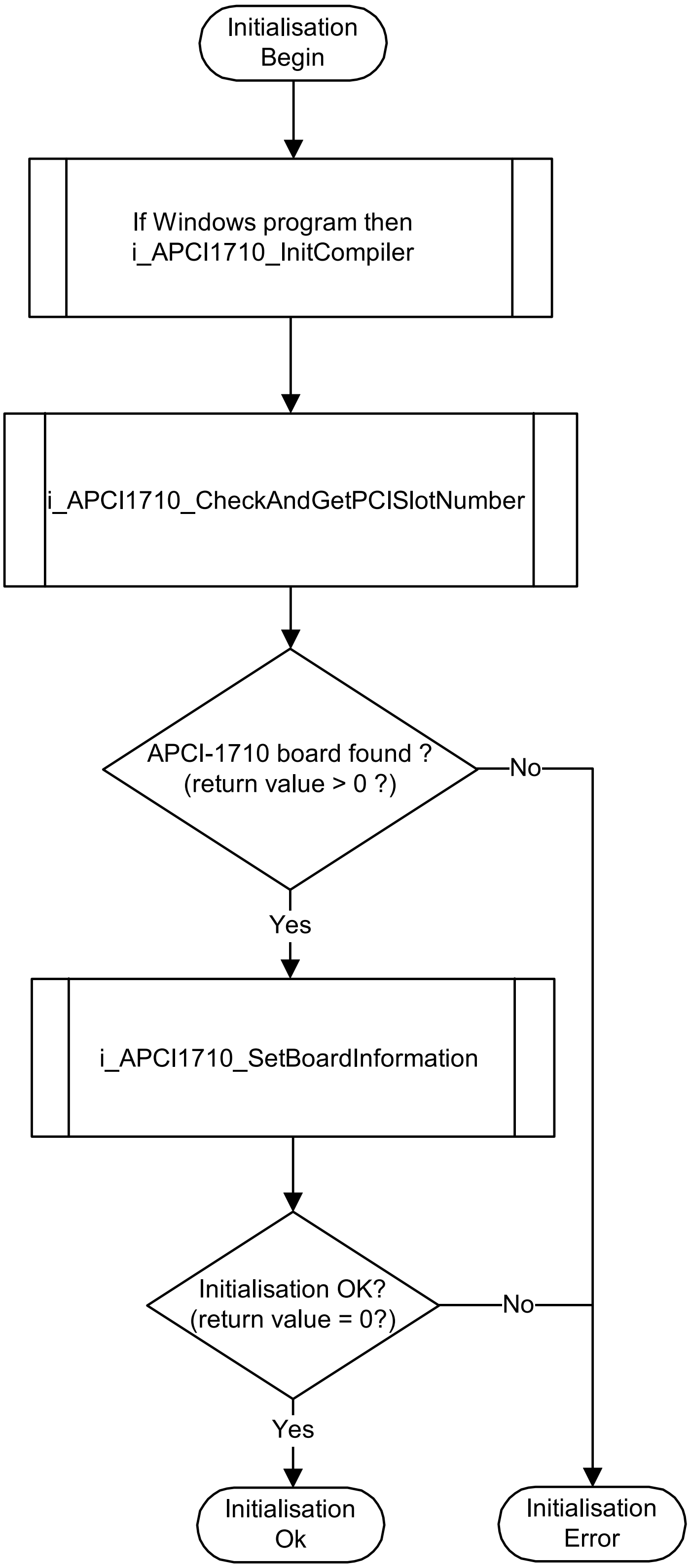
<Return Value>

4 SOFTWARE EXAMPLES

4.1 Initialisation

4.1.1 Initialisation of one APCI-/CPCI-1710board

a) Flow chart

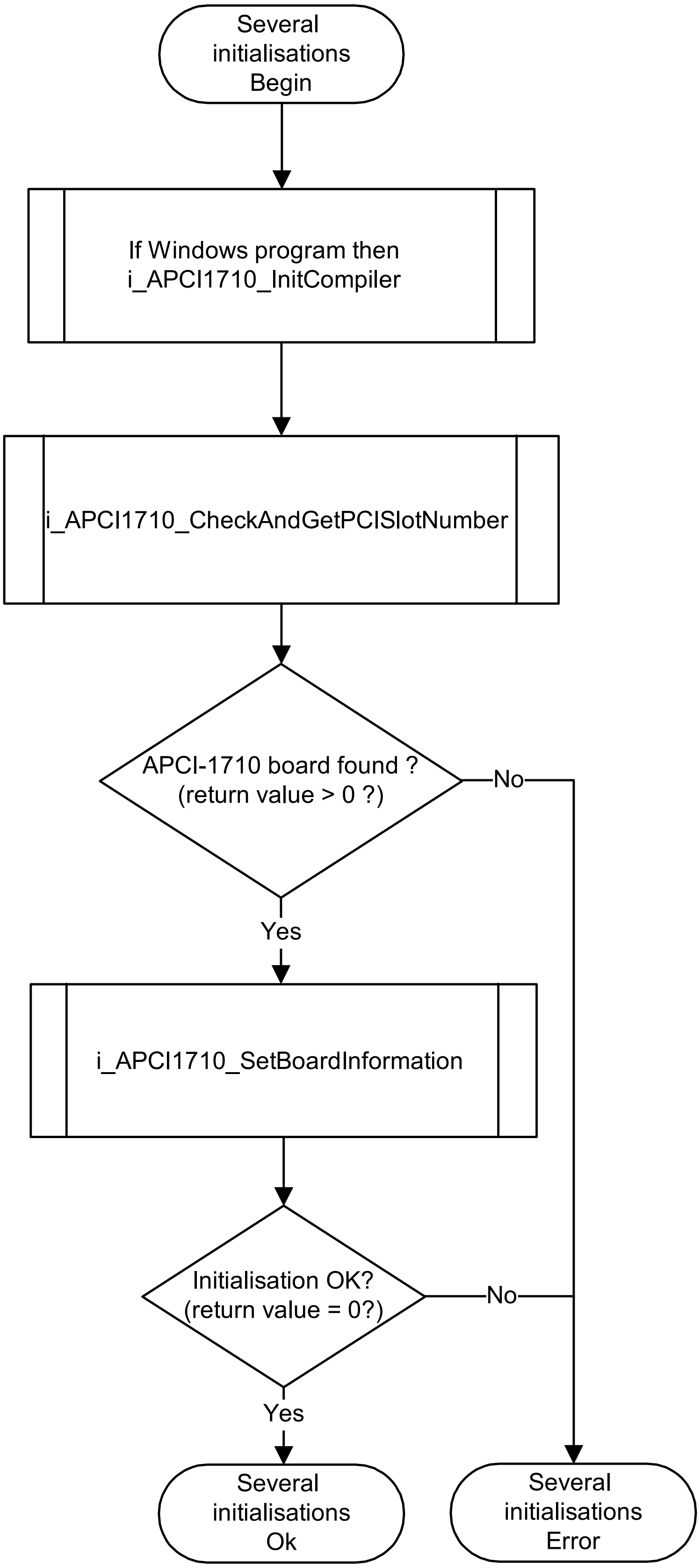


b) Example in C

```
int Initialisation(unsigned char *pb_BoardHandle)
    {
    unsigned char b_SlotNumberArray [8];

    #ifdef _Windows
       i_APCI1710_InitCompiler (DLL_COMPILER_C);
    #endif

    if(i_PCI1710_CheckAndGetPCISlotNumber (b_SlotNumberArray))
       {
       if(i_APCI1710_SetBoardInformation (b_SlotNumberArray[0],
                                          pb_BoardHandle) == 0)
          {
          return (0);      /* OK */
          }
       else
          {
          return (-1);     /* ERROR */
          }
       }
    else
       {
       return (-1);        /* ERROR */
       }
    }
```

4.1.2 Initialisation of several APCI-/CPCI-1710 boards

a) Flow chart



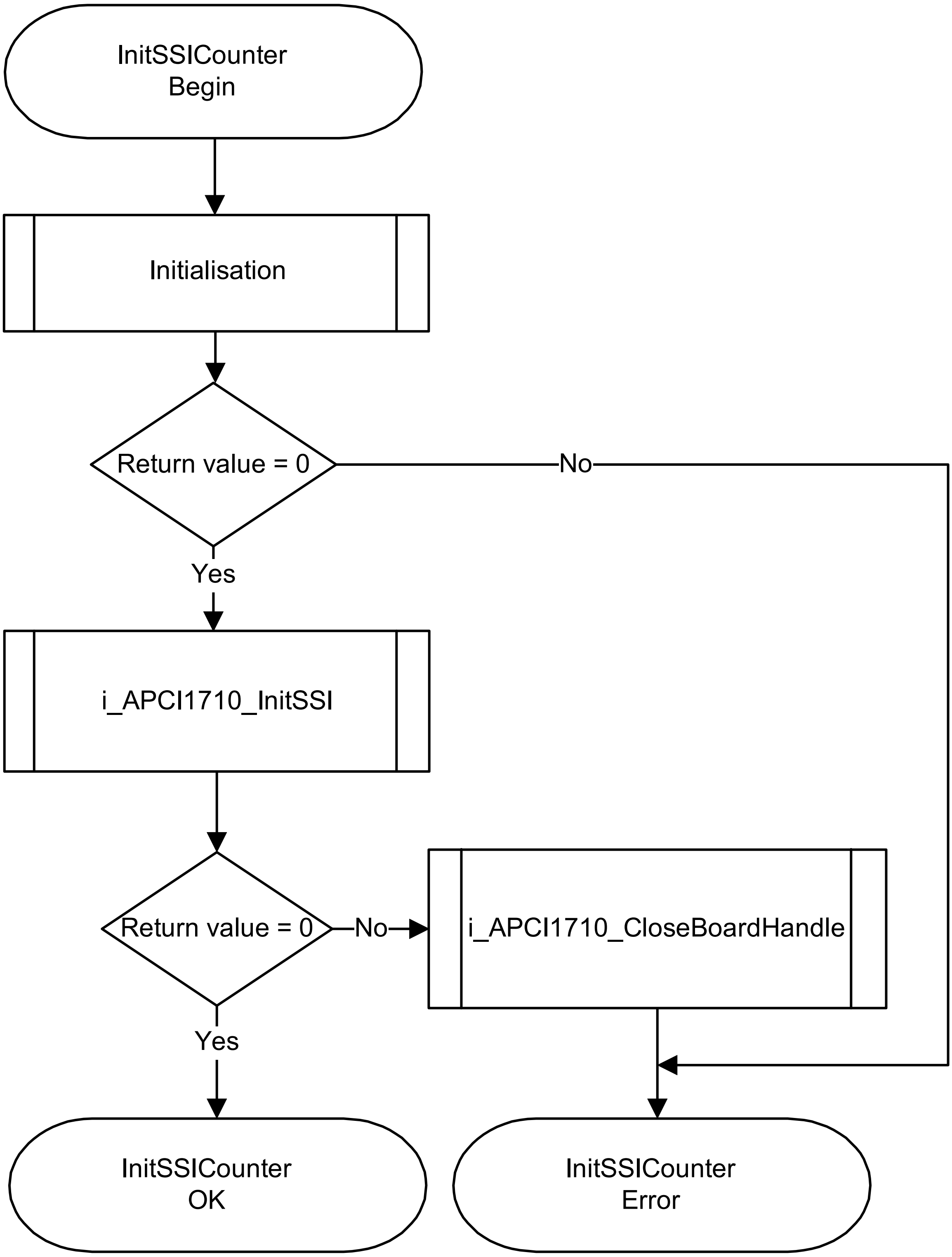
b) Example in C

```
int MoreInitialisation(unsigned char *pb_BoardHandleArray)
    {
    int           i_NbrOfBoard;
    int           i_Cpt;
    unsigned char b_SlotNumberArray [8];

    #ifdef _Windows
       i_APCI1710_InitCompiler (DLL_COMPILER_C);
    #endif

    i_NbrOfBoard = i_PCI1710_CheckAndGetPCISlotNumber (b_SlotNumberArray)

    if(i_NbrOfBoard > 0)
       {
       for (i_Cpt = 0; i_Cpt < i_NbrOfBoard; i_Cpt ++)
          {
          if (i_APCI1710_SetBoardInformation (b_SlotNumberArray[i_Cpt],
                                              &pb_BoardHandleArray [i_Cpt]) != 0)
             {
             break;
             }
          }

       if (i_Cpt == i_NbrOfBoard)
          {
          return (i_Cpt);  /* Return number of the found */
          }
       else
          {
          return (-1);     /* ERROR */
          }
       }
    else
       {
       return (-1);        /* ERROR */
       }
    }
```

4.2 SSI initialisation

4.2.1 Initialisation, 25-bit profile, 12-bit position length and 12-bit turn length

a) Flowchart



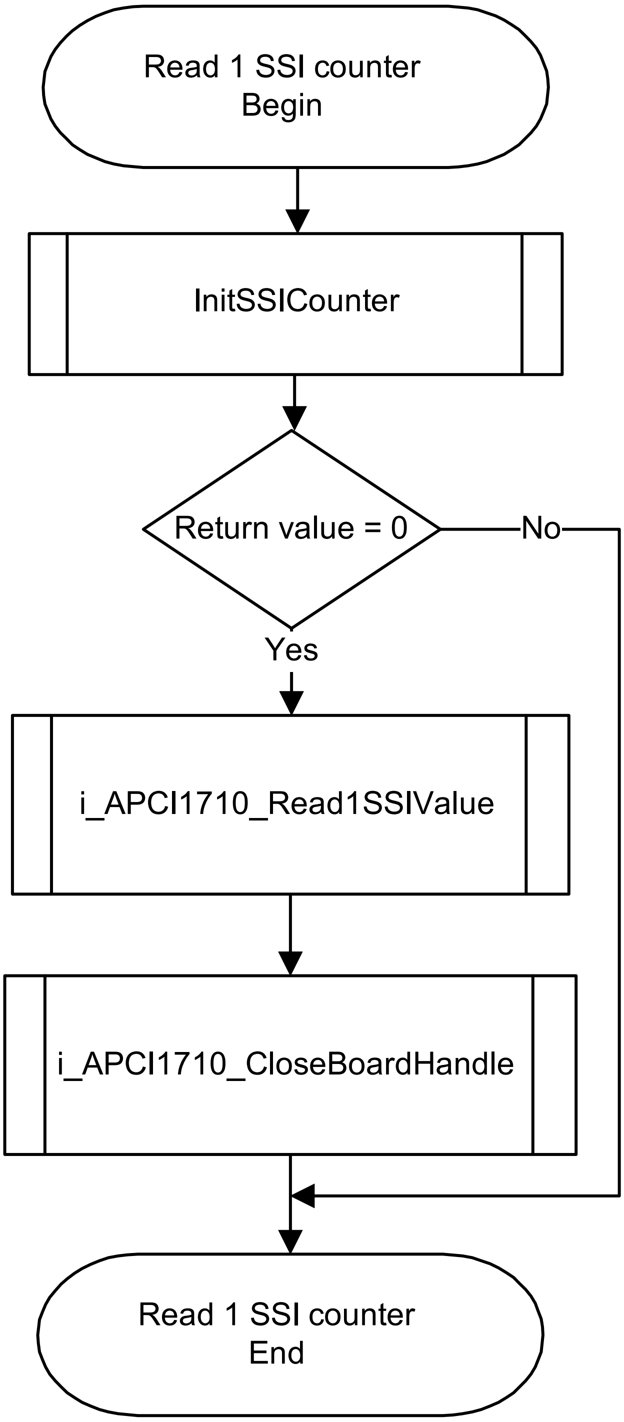
b) Example in C

```
int  InitSSICounter (unsigned char * pb_BoardHandle)
     {
     if (Initialisation (pb_BoardHandle) == 0)
        {
        if (i_APCI1710_InitSSI (pb_BoardHandle,
                                0,
                                25,
                                12,
                                12,
                                APCI1710_33MHZ,
                                150000,
                                APCI1710_BINARY_MODE) == 0)
           {
           printf ("Initialisation OK");
           return (0);
           }
        else
           {
           printf ("Initialisation error");
           i_APCI1710_CloseBoardHandle (*pb_BoardHandle);
           return (-1);
           }
        }
     else
        {
        printf ("Initialisation error");
        return (-1);
        }
     }
```

4.3 Reading SSI

4.3.1 Reading one SSI counter

a) Flow chart

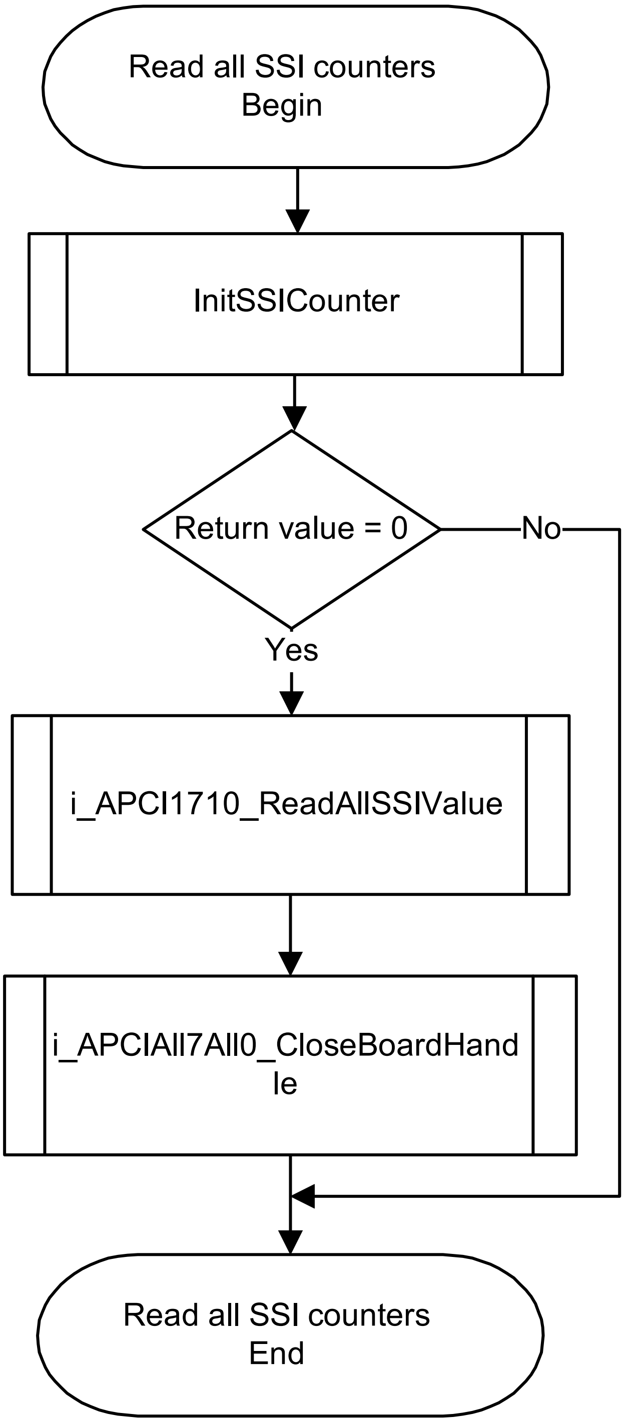


b) Example in C

```
void main (void)
     {
     int            i_ReturnValue;
     unsigned char  b_BoardHandle;
     unsigned long  ul_Position;
     unsigned long  ul_TurnCpt;

     if (InitSSICounter (&b_BoardHandle) == 0)
        {
        i_ReturnValue = i_APCI1710_Read1SSIValue (b_BoardHandle, 0, 0, &ul_Position,
&ul_TurnCpt);
        printf ("\nRead SSI counter return value = %d", i_ReturnValue);
        printf ("\nSSI position = %lu", ul_Position);
        printf ("\nSSI counting = %lu", ul_TurnCpt);

        i_APCI1710_CloseBoardHandle (b_BoardHandle);
        printf ("\nClose board handle return value = %d", i_ReturnValue);
        }
     else
        {
        printf ("Initialisation error");
        }
     }
```

4.3.2 Reading all SSI counters

a) Flow chart



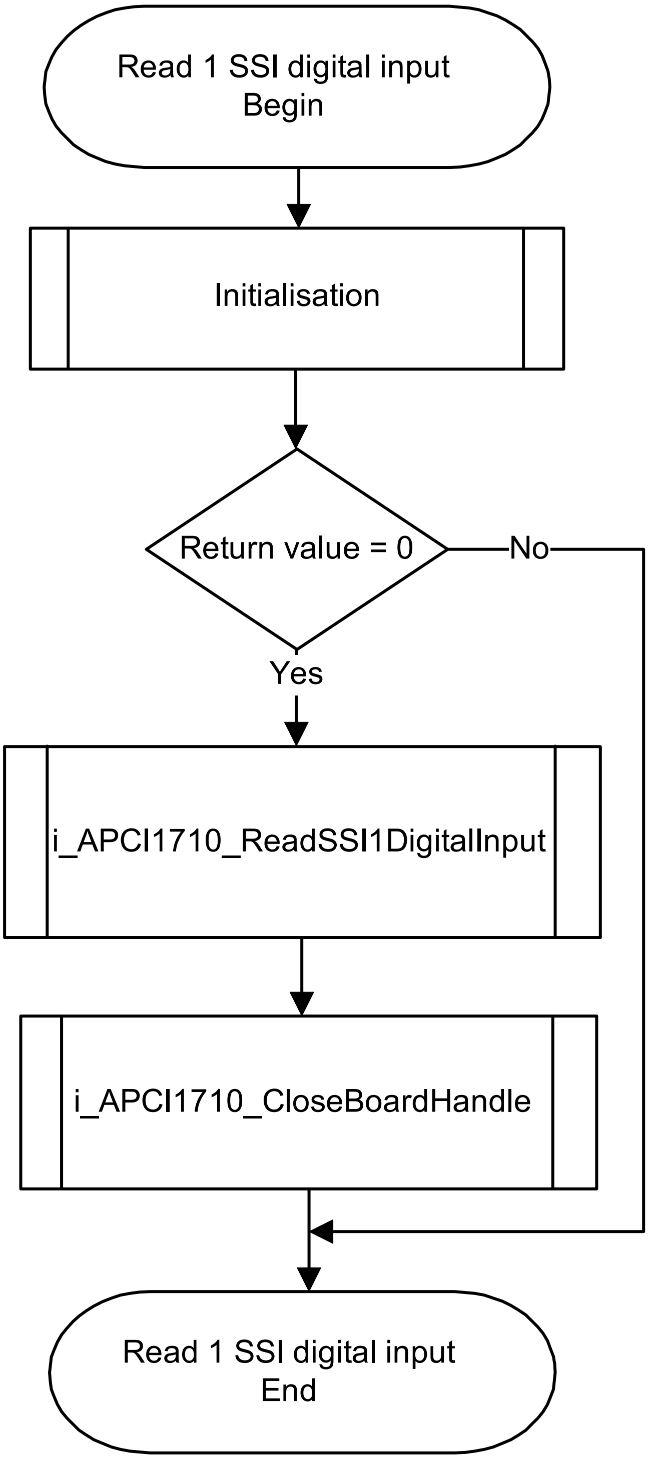
b) Example in C

```
void main (void)
     {
     int            i_ReturnValue;
     unsigned char  b_BoardHandle;
     unsigned long  ul_Position [3];
     unsigned long  ul_TurnCpt  [3];

     if (InitSSICounter (&b_BoardHandle) == 0)
        {
        i_ReturnValue = i_APCI1710_ReadAllSSIValue (b_BoardHandle, 0, ul_Position,
ul_TurnCpt);
        printf ("\nRead SSI counter return value = %d", i_ReturnValue);
        printf ("\nSSI counter 0 position = %lu", ul_Position [0]);
        printf ("\nSSI counter 0 counting = %lu", ul_TurnCpt  [0]);
        printf ("\nSSI counter 1 position = %lu", ul_Position [1]);
        printf ("\nSSI counter 1 counting = %lu", ul_TurnCpt  [1]);
        printf ("\nSSI counter 2 position = %lu", ul_Position [2]);
        printf ("\nSSI counter 2 counting = %lu", ul_TurnCpt  [2]);

        i_APCI1710_CloseBoardHandle (b_BoardHandle);
        printf ("\nClose board handle return value = %d", i_ReturnValue);
        }
     else
        {
        printf ("Initialisation error");
        }
     }
```

4.4 SSI digital input channels

4.4.1 Reading one digital input channel

a) Flow chart

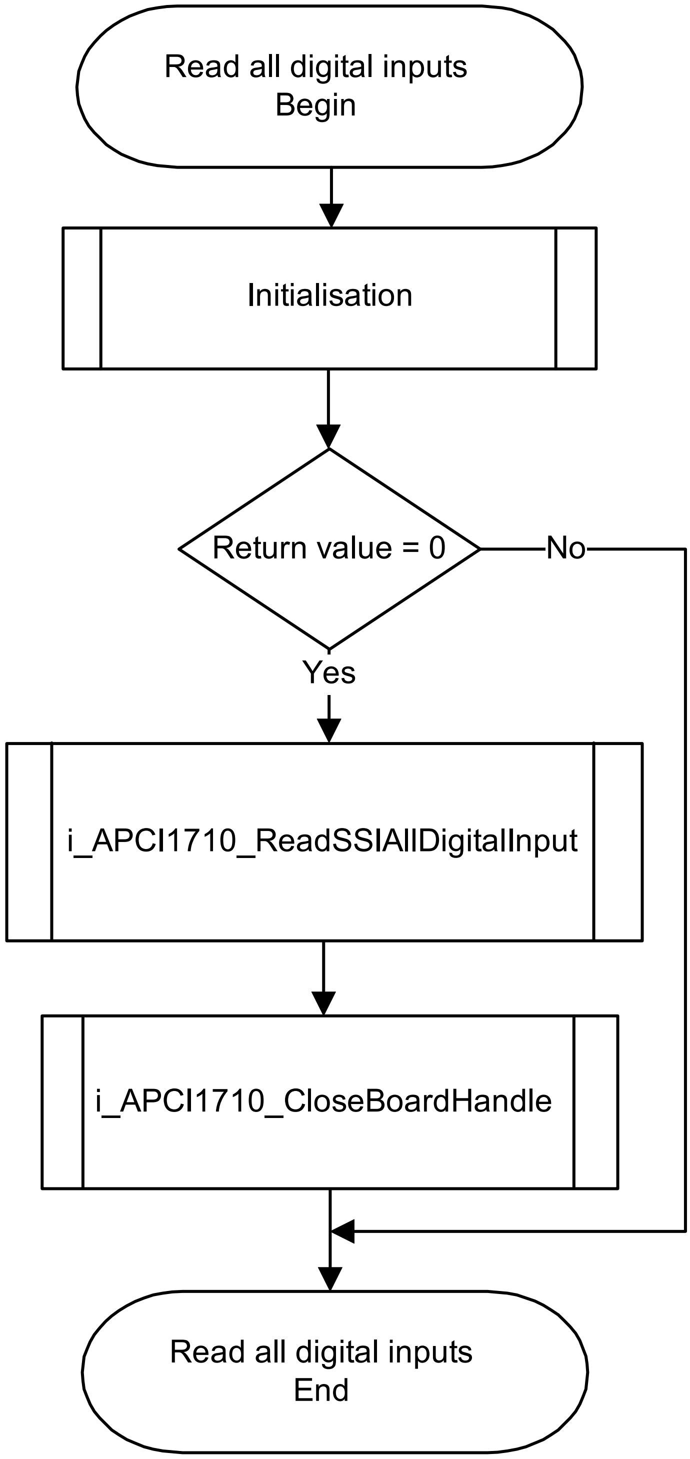


b) Example in C

```
void main (void)
     {
     int           i_ReturnValue;
     unsigned charb_BoardHandle;
     unsigned charb_ChannelStatus;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        i_ReturnValue = i_APCI1710_ReadSSI1DigitalInput(b_BoardHandle, 0, 0,
&b_ChannelStatus);
        printf ("\nRead 1 SSI digital input return value = %d", i_ReturnValue);
        printf ("\nSSI digital input status = %d", b_ChannelStatus);

        i_APCI1710_CloseBoardHandle (b_BoardHandle);
        printf ("\nClose board handle return value = %d", i_ReturnValue);
        }
     else
        {
        printf ("Initialisation error");
        }
     }
```

4.4.2 Reading all digital input channels

a) Flow chart



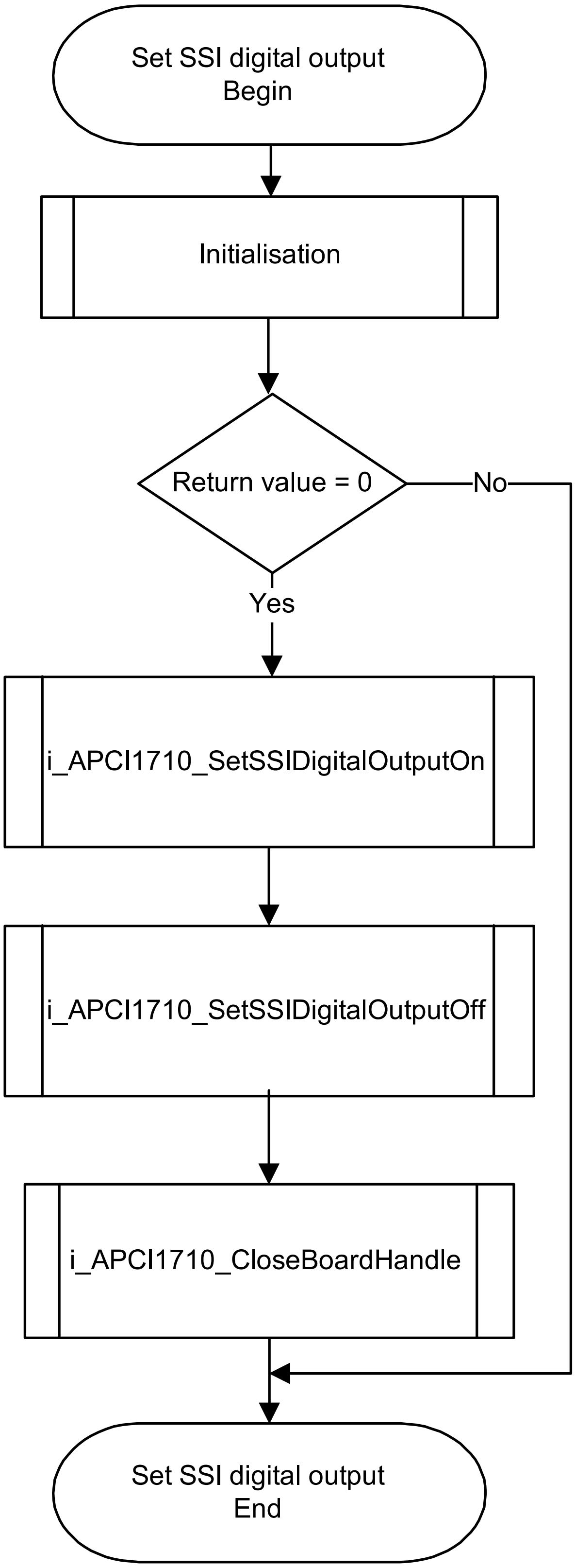
b) Example in C

```
void main (void)
     {
     int           i_ReturnValue;
     unsigned charb_BoardHandle;
     unsigned charb_InputStatus;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        i_ReturnValue = i_APCI1710_ReadSSIAllDigitalInput(b_BoardHandle, 0,
&b_InputStatus);
        printf ("\nRead all SSI digital input return value = %d", i_ReturnValue);
        printf ("\nSSI digital input status = %d", b_InputStatus);

        i_APCI1710_CloseBoardHandle (b_BoardHandle);
        printf ("\nClose board handle return value = %d", i_ReturnValue);
        }
     else
        {
        printf ("Initialisation error");
        }
     }
```

4.5 SSI digital output channel

4.5.1 Setting the digital output channel

a) Flow chart



b) Example in C

```
void main (void)
     {
     int          i_ReturnValue;
     unsigned charb_BoardHandle;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        i_ReturnValue = i_APCI1710_SetSSIDigitalOutputOn(b_BoardHandle, 0);
        printf ("\nRead set SSI digital output on return value = %d",
i_ReturnValue);

        i_ReturnValue = i_APCI1710_SetSSIDigitalOutputOff(b_BoardHandle, 0);
        printf ("\nRead set SSI digital output off return value = %d",
i_ReturnValue);

        i_APCI1710_CloseBoardHandle (b_BoardHandle);
        printf ("\nClose board handle return value = %d", i_ReturnValue);
        }
     else
        {
        printf ("Initialisation error");
        }
     }
```