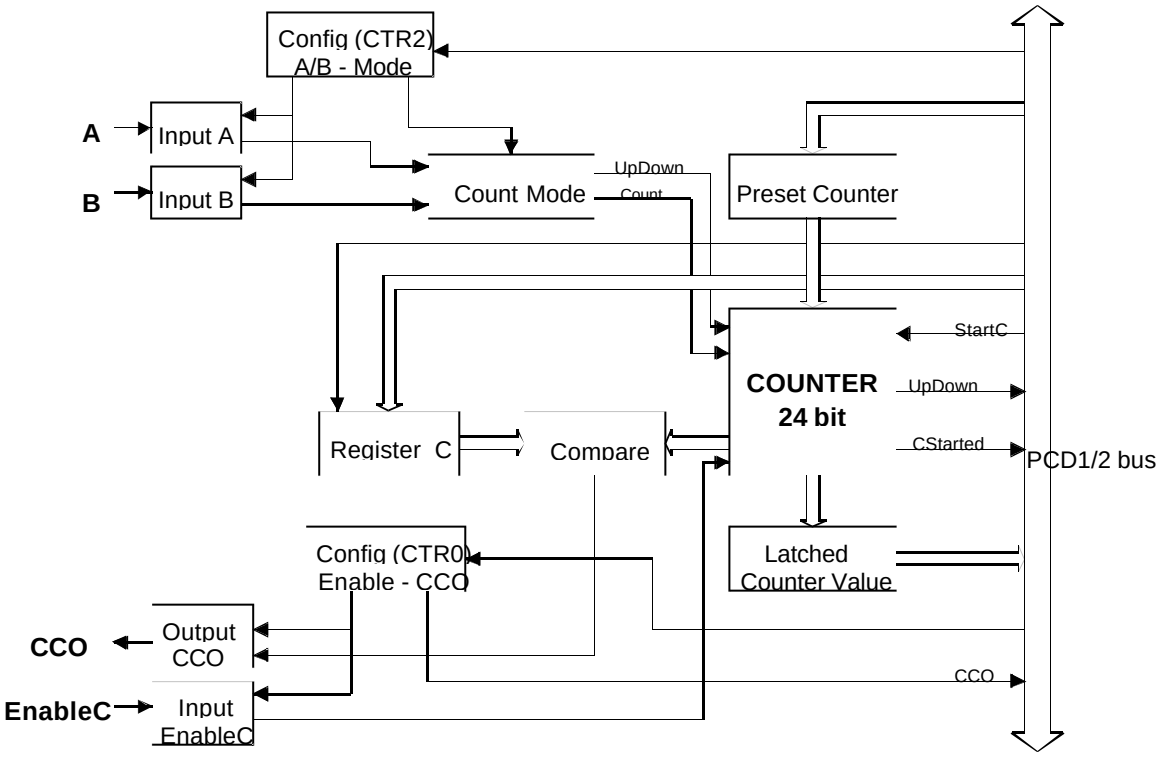
3 PCD2.H110 and fast counting mode

Block diagram of counter and description



Inputs as “A”,”B” and “Enable” as well as the Output “CCO” have to be configured (this is described in a later section).

The core of the circuit is a 24 bit counter. A preset value for the counter can be loaded from the Preset Counter Register. The Preset Counter Register itself is loaded by the user program with direct periphery access command (**LoadP**). The "**StartC**" command loads the preset counter value into the counter and starts it. Input “A” in conjunction with input “B” Inc-Decrement the counter value.

The **Register_C** is loaded in a similar way. The user program use direct periphery access. When the counter reach the value of the compare register (Register_C), the CCO will be switched according to its configuration so that the process and the user program can be controlled.
The status of "CCO", "Cstarted” and "UpDown" can be read by the user program with direct periphery access.

The user program can read the current counter value from the "Latched Counter Value" buffer. The counter can be read during counting. This reading is processed by direct periphery access. It is important to be aware that the value read during counting always arrives with some delay. Only readings

obtained after counting has finished can be correctly evaluated. For comparisons, always use the CCO output in combination with the Register_C.

3.1 Configuration of the PCD2.H110 for counting

3.1.1 Hardware configuration

The Module has to be declare in the hardware configuration of the PCD.xx7, this is done on the DB1, DB511 or DB1023.

The module identification is : 81h
Number of input byte needed are : 8
Number of output byte needed are : 14

In This module has to be out of the process image area, so the Input and Output address have to be over the address 256 (e.g. CPU 414).

In this example the module PCD2.H110 is on the first slot and its base address are 700 for input and output.

Address	Name	Type	Initial Value	Comment
0.0		STRUCT		
+0.0	Kennbyte1	CHAR	'M'	This identify this DB
+1.0	Kennbyte2	CHAR	'x'	as the hardware configuration DB,
+2.0	Kennbyte3	CHAR	'x'	and the module setting.
+3.0	Kennbyte4	CHAR	'7'	
+4.0	Modul21	STRUCT		
+0.0	kenn	WORD	W#16#81	
+2.0	PANr	INT	0	
+4.0	InCnt	INT	8	
+6.0	OutCnt	INT	14	
+8.0	InBase	INT	700	
+10.0	OutBase	INT	700	
+12.0	mask	BYTE	B#16#0	
+13.0	dummy_b	BYTE	B#16#0	
+14.0	INIT_COUNT	BYTE	B#16#0	
+15.0	INIT_MEASURE	BYTE	B#16#0	
=16.0		END_STRUCT		

You can also see, that there are two byte parameters in the structure , which are INIT_COUNT and INIT_MEASURE. For the counter function we need only to configure the parameter INIT_COUNT. The parameter INIT_MEASURE will be seen in the chapter for measurement functions.

3.1.2 Parameter INIT_COUNT

This parameter is a byte, each bit of this byte has his own meaning.

Bit N°	Description
0	1 = Input EnableC is Dynamic 0 = Input EnableC is Static
1	1 = Input EnableC is Inverted 0 = Input EnableC is not inverted
2	1 = Output CCO is Static 0 = Output CCO is Dynamic
3	1 = Output CCO is inverted 0 = Output CCO is not inverted
4	1 = Counter input ”A” is inverted 0 = Counter input “A” is not inverted
5	1 = Counter input ”B” is inverted 0 = Counter input “B” is not inverted
7..6	00 = Counting MODE x1 01 = Counting MODE x2 10 = Counting MODE x3 11 = Counting MODE x4

Let0s see those parameter in details :

3.1.2.1 EnableC input

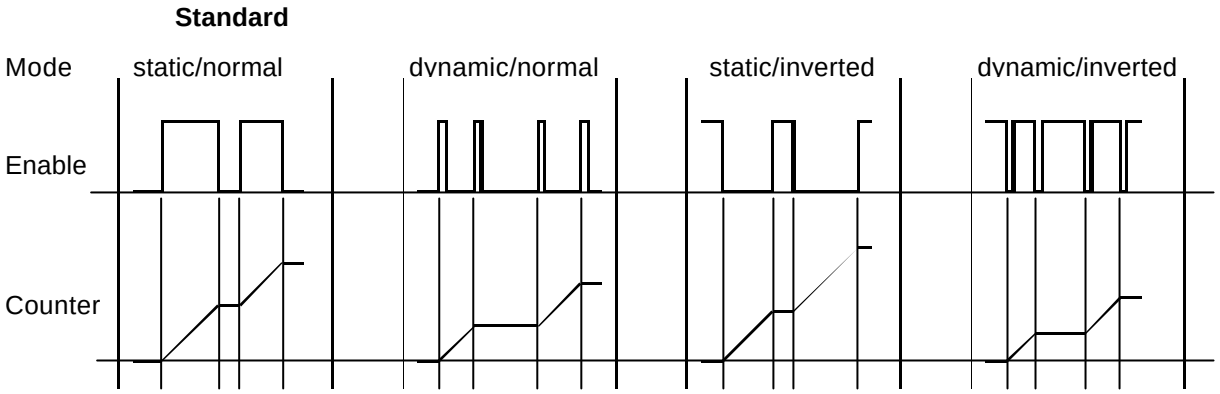
Standard: "static / normal"
While the Enable input is "H", counting is allowed.
While the Enable input is "L", counting is stopped.

Additional possibilities:

"static / inverted"
While the Enable input is "L", counting is allowed.
While the Enable input is "H", counting is stopped.

"dynamic / normal"
The Enable input is "L". The first positive edge (H) switches Enable on, the next switches it off again, etc.

"dynamic / inverted"
The Enable input is "H". The first negative edge (L) switches Enable on, the next switches it off again, etc.



3.1.2.2 CCO output

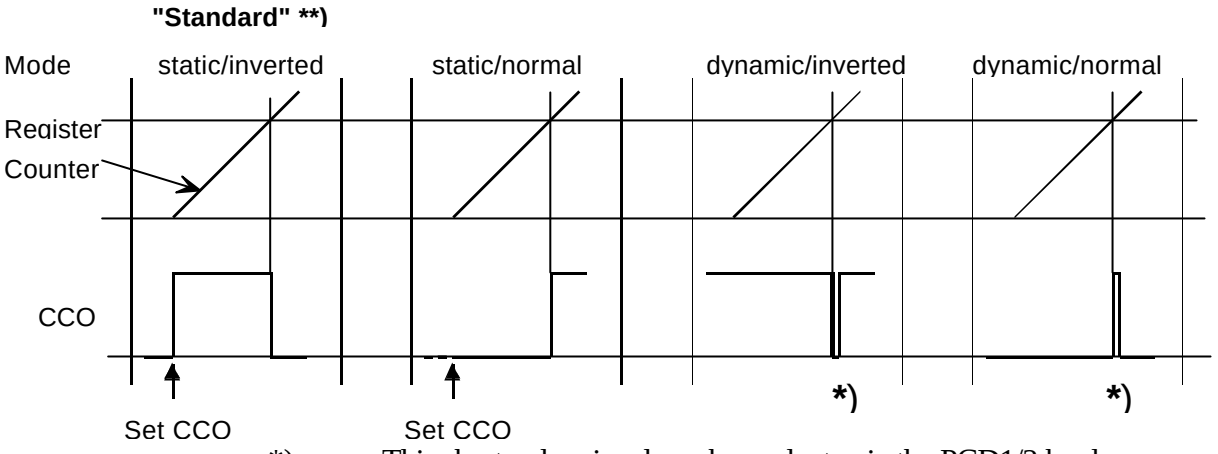
Standard: "static / inverted"
The CCO is switched on by the user program and becomes H. If the register and counter are equal, the CCO is switched L and remains L until a new command to switch on is received from the user program.

Additional possibilities:

"static / normal"
The CCO is activated by the user program and becomes or remains L. If the register and counter are equal, the CCO is switched H and remains H until a new activate command is received from the user program.

"dynamic / inverted"
The CCO is switched on by the user program and becomes H. If the register and counter are equal, the CCO becomes L *) for 25 .. 100 µs. At each subsequent agreement of register and counter, the behavior of the CCO is repeated, without any new instructions from the user program.

"dynamic / normal"
The CCO is activated by the user program and becomes or remains L. If the register and counter are equal, the CCO becomes H *) for 25 .. 100 µs. At each subsequent agreement of register and counter, the behavior of the CCO is repeated, without any new instructions from the user program.



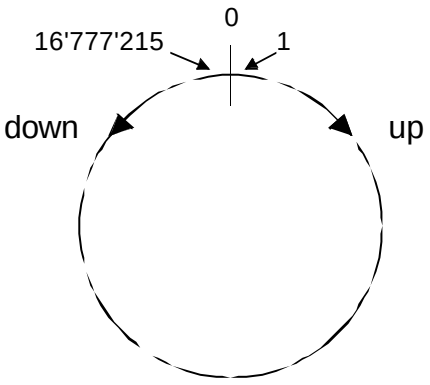
*) This short pulse signal can be evaluate via the PCD1/2 hardware input interrupt and the OB 40.

**) "Standard" indicates the mode, which is probably used most often in practice.

3.1.2.3 Counting Mode x

Mode "x1" (without incremental shaft encoder) is used for simple counting tasks:

- The signals to be counted are at input A
- If input B is L, count direction is downwards
  If input B is H, count direction is upwards



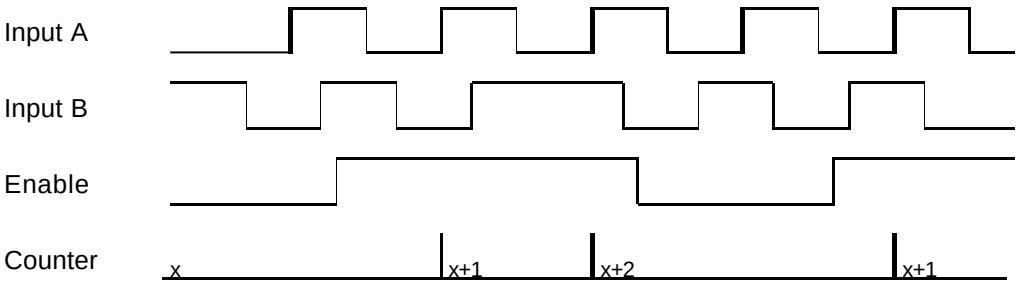
Counting range is 0 ... 16'777'215 (0 ... 2^{24} - 1)

If up-counting from 0 → 0, 1, 2 ...
If down-counting from 0 → 0, 16'777'215, 16'777'214 ...

There are no negative values and no overflow

If 2-phase incremental shaft encoders are used, the following modes are available:

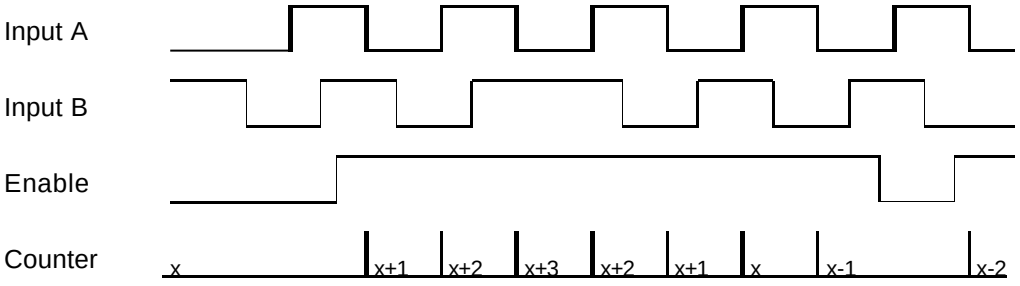
Mode "x1"



Only the rising edge of signal A is evaluated. Signal B in quadrature (phase shifted by 90°) defines the count direction.

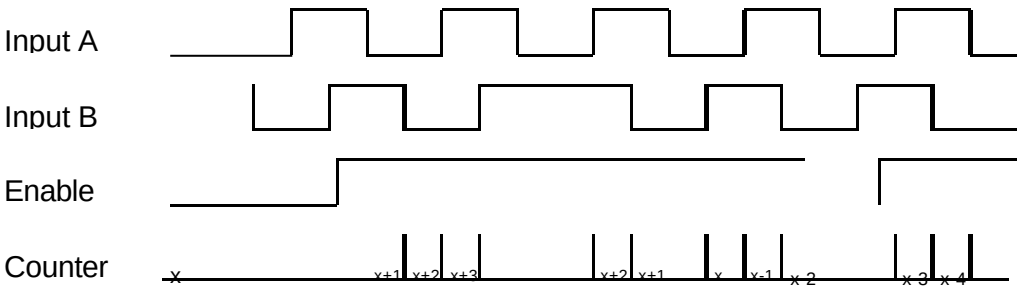
Mode "x2"

The rising and falling edges of signal A are evaluated. Signal B in quadrature defines the count direction.



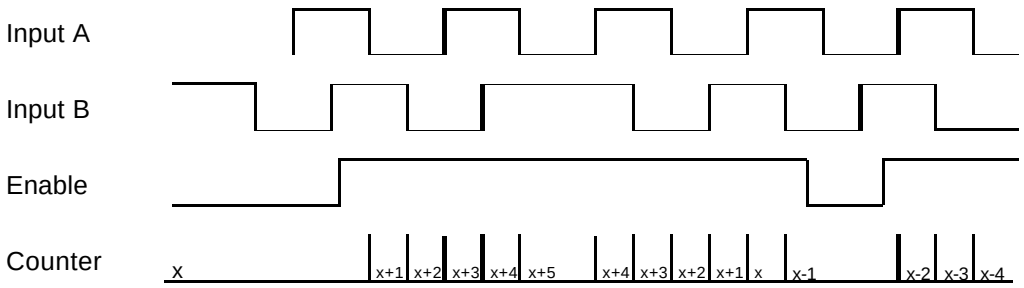
Mode "x3"

The rising and falling edges of signal A are evaluated and also the rising edge of signal B. Signal B in quadrature defines the count direction.



Mode "x4"

The rising and falling edges of both signals A and B are evaluated. Signal B in quadrature again defines count direction.



3.1.3 Configuration example

If we like to have the input "A" inverted, Input "B" not inverted and the Input EnableC passive. The "CCO" Output Static inverted and the mode count X2 . The parameter INIT_COUNT will take the value = b#16#5C = 01011100

3.2 Programming function for the counter access

Access to the Module function is done through out direct periphery access. Programming is explained on the following pages, using a few examples.

3.2.1 Reading function

Offset	Use	Operation	Description
+ 0.0	C	L PED	Read the actual counter value
+ 4.0	M	L PEW	Reserved for measurement mode
+ 6.0	C	L PEB	Counter Status Bit 0 = UpDown status bit (1 = up) Bit 1 = Status of the CCO Bit 2 = Status of the Cstart (1= started)
+ 7.0	M	L PEB	Reserved for measurement mode

3.2.2 Writing function

Offset	Use	Operation	Description
+ 0.0	C	T PAD	Transfer to the module the Preset Counter value
+ 4.0	C	T PAD	Transfer to the Register_C the compare value
+ 8.0	M	T PAW	Reserved for measurement mode
+ 10.0	C	T PAB	Start the counter
+ 11.0	M	T PAB	Reserved for measurement mode
+ 12.0	M	T PAB	Reserved for measurement mode

Function for Measurement part of the module, see chapter 2.5.

3.2.3 Example of programming

In this example we will configure the input output this way:
"A" and "B" inputs are not inverted.
"EnableC" is Dynamic and not inverted.
"CCO" is Static and not inverted.
This give us the value **b#16#45** fotr the parameter INIT_COUNT for the DB1 (Hardware configuration DB).
The base address selected for the input and output is **700** .

In this simple example, when a rising edge of the input E2.0 has been detected, we load the compare and preset value into the module and then start the counter.

```
U     E      2.0
FP    M    200.0
SPBN  stx1
L     100
T     PAD  700 // Preset counter value

L     300
T     PAD  704        // Compare value into Register_C

T     PAB  710        // Start counter

Stx1: NOP 0
```

Remark: Of course the counting will really start when a rising edge is detected on the EnableC module input.

Then we watch when the Compare value is reached, we do it looking the CCO status.

```
L     PEB  706        // load status byte into ACCU1
T       MB   201  // Transfer into a byte Flag

U       M    201.1  // check the "CCO" bit flag
=       M    204.0  //put the result in a bit
flag
```

When the CCO is set, then we reload the counter and restart it. This is done just by adding a condition in the starting procedure.

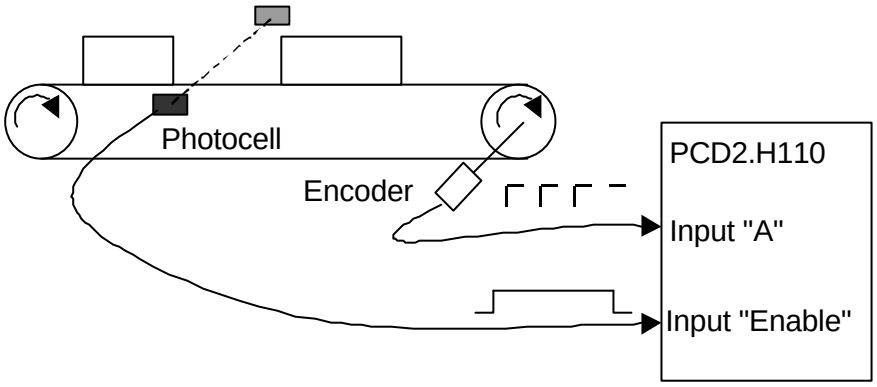
```
U     E      2.0
FP    M    200.0
O     M    204.0
SPBN  stx1
```

The full version of the example can be found in the attached project DOC_H110.ARJ, the FC1. To run it you need first to edit the OB100 to set the correct mode.

3.3 Application example with the counter

Object are carried on a conveyor belt, when those object cross a photoelectric barrier, the signal going out of the photoelectric cell will be proportional to the **size of the object** and to the speed of the conveyor belt. But also the size of the object on the conveyor belt will be proportional to the distance covered by the conveyor belt.
So it possible then to find out which are the size of the object crossing the photocell. This simple method has been successfully used in the south of France to sort melons and apricots.

For this purpose, an encoder is fixed on the conveyor, which will give us the covered distance by pulse, so we connect it to the input ”A”. The photocell will be connected to the input EnableC, which will valid the measurement time (1 = Object seen).



3.3.1 Configuration of the module Input/Output

- The “EnableC” will be use in a static way and not inverted, when the signal is High it counts then when the signal is low it stops to count.
- The output “CCO” is not use in this application, so the configuration of it is irrelevant.
- The counting mode will be the mode x1, because no signal will be available for the input “B”.
- “A” and “B” input are not inverted, this is irrelevant.

This give us the parameter for the hardware configuration DB, INIT_COUNT = **b#16#0** .

3.3.2 The software

We need to initialize the module in two case.
1) In the first cycle to prepare the module.
2) Between each measurement to reset the counter value

To do so we will use a flag M150.0, when this flag is set we initialize the module. In the init we will reset the counter value, set a high value for the compare register (because compare function is not needed) and start the counter , for the counter to be ready when the EnableC signal is coming.

So the code look like that:

```
U     M    150.0                 // Flag to init the counter
SPBN  m001
R     M    150.0                 // reset the init flag


L     0                          // Preset value is zero
T     PAD  700                   // always start counting from zero


//****  Compare function is not needed, so we load a high //****  value which will never be
reached,
//****  to not disturb our application

L     dw#16#7FFFFF
T     PAD  704


T     PAB  710                   // Start counter

m001: NOP   0
```

Now we need to find out when the measurement is completed, to do so we will check the Status bit Cstart.
When Cstart is = 1, this means that the module is counting, when is going down to 0 it means that the EnableC signal disappeared.
So when the count is stopped we will read the value of the counter and set the flag to init the module for the next measurement.

```
L     PEB  706                   // load the status counter byte
T     MB   152                   // transfer the copy in MB152

U     M    152.2                 // Check the bit CStart
FN    M    150.2                 // for a falling edge
SPBN  m002                       // if no falling edge jump to m002

S     M    150.0                 // set the flag for new init
S     #NEW_Value                 // set the flag new_Data_ready

L     PED  700                   // load the value in the counter
T     #SIZE                      // put it in a output variable


m002: NOP   0
```

