PRELIMINARY

The synchronizer pipe bloc

General description

The synchronizer has been designed to provide a means to switch on and off a sub-pipe. Its two main states are SYNCHRONIZED and UNSYNCHRONIZED (ON and OFF respectively). In order to insure that the transitions between these two states can be achieved without discontinuities on the position and its derivatives, the synchronizer provides two type of ramp: one based on a parabolic curve and an other based on a 3-4-5 polynomial.

In addition to the type of the curve, the size of the synchronization must be specified. This size called Δs or phasing value is expressed as the distance travelled by the slave while going from one state, SYNCHRONIZED or UNSYNCHRONIZED, to the other.

Declaration

```
SYNCHRONIZER <block_name> ;
     CURVE_TYPE      = [ PARABOLA | POLYNOMIAL_3_4_5 ] ;
     OUTPUT_PHASING  = <phasing_value> ;
END
```

Where

CURVE_TYPE specifies the transfer function used during transitions.
<phasing_value> corresponds to Δs, specifies the distance travelled by the slave during a transition. Can be any non-null real value.

Accessibility

CURVE_TYPE is not accessible.
OUTPUT_PHASING can be read and written.

Note that it makes no sense to connect an auxiliary input to the OUTPUT_PHASING field. So this syntax is not allowed.

Sign of <phasing_value>

The sign of <phasing_value> indicates if the synchronizer works in a system where the input is increasing (sign of <phasing_value> is positive) or decreasing (negative). It is thus very important to know the actual direction of rotation at the time of the execution of a start or stop command. For example, to successfully achieve a start or stop command while input is increasing then the sign of <phasing_value> must be positive. Inversely, to successfully achieve a start or stop command while input is decreasing, the sign of <phasing_value> must be negative. On the other hand, trying to start or stop while the input grows in the opposite direction of the one defined by the sign of <phasing_value> the synchronizer will never be able to synchronize or desynchronize.

Commands

```
<block_name> <- start ;
<block_name> <- stop(<rest_value>) ;
<block_name> ? ready ;
<block_name> ? status ;                  // return the synchronizer status word
<block_name> ? status(<status_mask>) ;   // return a boolean
```

Where

`<rest_value>` specify the stop position of the slave. Can be any real number.
“`<block_name> ? ready`” is true if the last command has been completed and false otherwise.
`<status_mask>` Word used to mask the synchronizer status.

There are two predefined masks available in samerror.sys :
SYN_SYNCHRONIZED : TRUE when the master is equal to the slave.
SYN_RAMP_ON : TRUE when the slave is driven by the generator.

Synchronization curves

Here are the two type of curve available:

`PARABOLA` : $s' = 2m'^2$
`POLYNOMIAL_3_4_5` : $s' = 10\ m'^3 - 15\ m'^4 + 6\ m'^5$
where s´ and m´ are the instantaneous normalised output (slave) and respectively input (master) values.

Both laws are described in details in "VDI 2143 Blatt 1: Bewegungsgesetze für Kurvengetriebe, Theoretische Grundlagen" and "VDI 2143 Blatt 2: Bewegungsgesetze für Kurvengetriebe, Praktische Anwendung".

Basically, the first law is simpler and allows a minimal acceleration. Its disadvantage is that the jerk has peaks at both ends (the acceleration goes suddenly from 0 to constant value and again to 0 at the end). The second law is slightly more complicated but it allows a controlled jerk. On the other hand, the peak acceleration is quite larger than with the first law and the master displacement during this operation is a little bit shorter.

Here is a detailed table of basic properties of both laws:

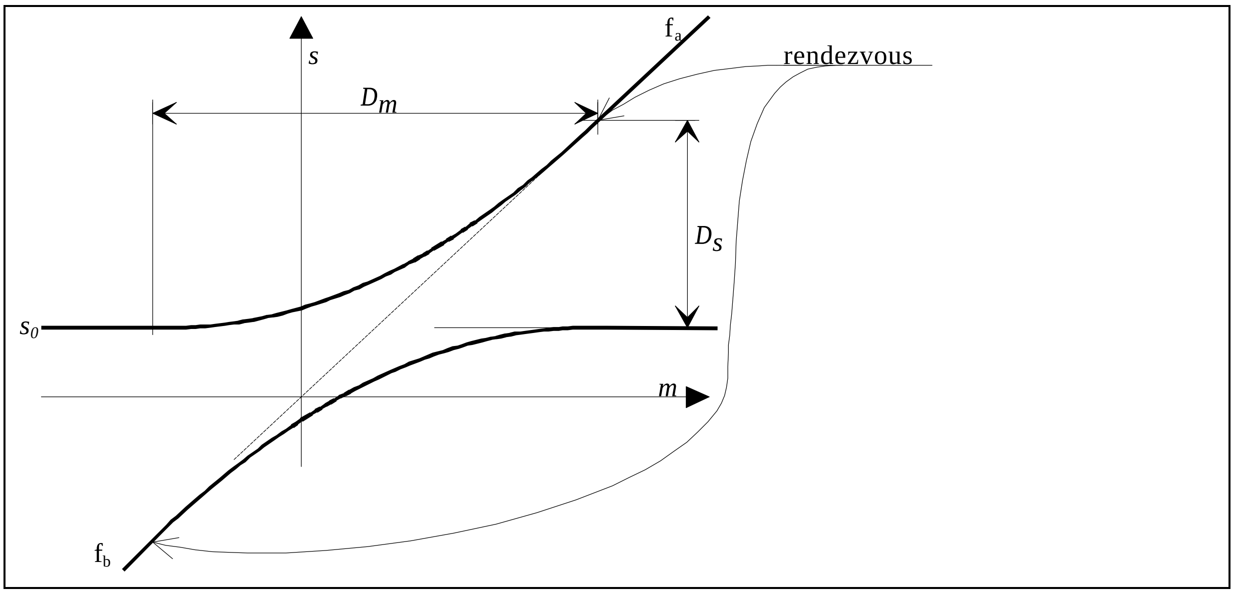
value / kind of law	parabola	3-4-5 polynomial	$\frac{\text{3-4-5polynomial}}{\text{parabola}}$
master position at beginning of phasing in (or out)	$m_b = s_0 - \Delta s$	$m_b = s_0 - \frac{7}{8}\Delta s$	---
master position at end of phasing in (or out)	$m_e = s_0 + \Delta s$	$m_e = s_0 + \Delta s$	---
master distance to phase in/out	$\Delta m = 2\Delta s$	$\Delta m = \frac{15}{8}\Delta s$	$\frac{15}{16} \cong 0.9375$
Acceleration (master at constant speed)	$\lvert\hat{\ddot{s}}\rvert = \frac{\dot{m}^2}{2\Delta s}$	$\lvert\hat{\ddot{s}}\rvert = \frac{\dot{m}^2}{2\Delta s}\cdot\frac{128}{135}\cdot\sqrt{3}$	$\frac{128}{135}\cdot\sqrt{3} \cong 1.642$
Jerk (master at constant speed)	$\lvert\hat{\dddot{s}}\rvert = (\infty)$	$\lvert\hat{\dddot{s}}\rvert = \frac{\lvert\dot{m}\rvert^3}{\Delta s^2}\cdot\frac{512}{225}$	---
with: $\dot{m} = \frac{dm}{dt}$ the (constant) speed of the master, Δs the slave distance to phase in (or out) and s_o the rest position of the slave			

Example

Suppose an UNSYNCHRONIZED synchronizer pipe block with its output being at its rest position s_0 the rendezvous will occur when both input and output values equal m_e. The output will begin « ramping » when the input value equals m_b. The sign in the table depends if the input values are increasing (upper one) or decreasing (lower one).

Graphical Representation

The following drawing shows in details the transfer function of the synchronizer pipe block while it is changing from on to off (start command) or conversely (stop command). The function f_a corresponds for an increasing input to a start or for a decreasing input to a stop command. Conversely the function f_b corresponds for an increasing input to a stop or for a decreasing input to a start command.
It is obvious that the transfer function when on (synchronized) is $s = m$ and its graphical representation is superimposed with the first bisector in a (m ;s) graph while the one when off (desynchronized) is $s = s_0$ and its graphical representation is a horizontal line at height s_0 in the same (m ;s) graph.



Detailed Description

The following state-transition diagram describes in details how the synchronizer treats start and stop commands as well as how it responds to activation and deactivation.

States

READY: In this state, the synchronizer is either SYNCHRONIZED or UNSYNCHRONIZED depending on the value of the S variable. The output value is either the rest position if UNSYNCHRONIZED or the input value if SYNCHRONIZED.

EXECUTING: The synchronizer is in this state from the moment it receives a new command until the end of the transition. In this state, the output value is computed by the ramp generator using the specified type of curve and Δs.

STACKED: The synchronizer is in this state while executing the first part of a double command. A double command is either a start-stop or a stop-start. In this state, the ramp generator as in EXECUTING computes the output value.

Variables

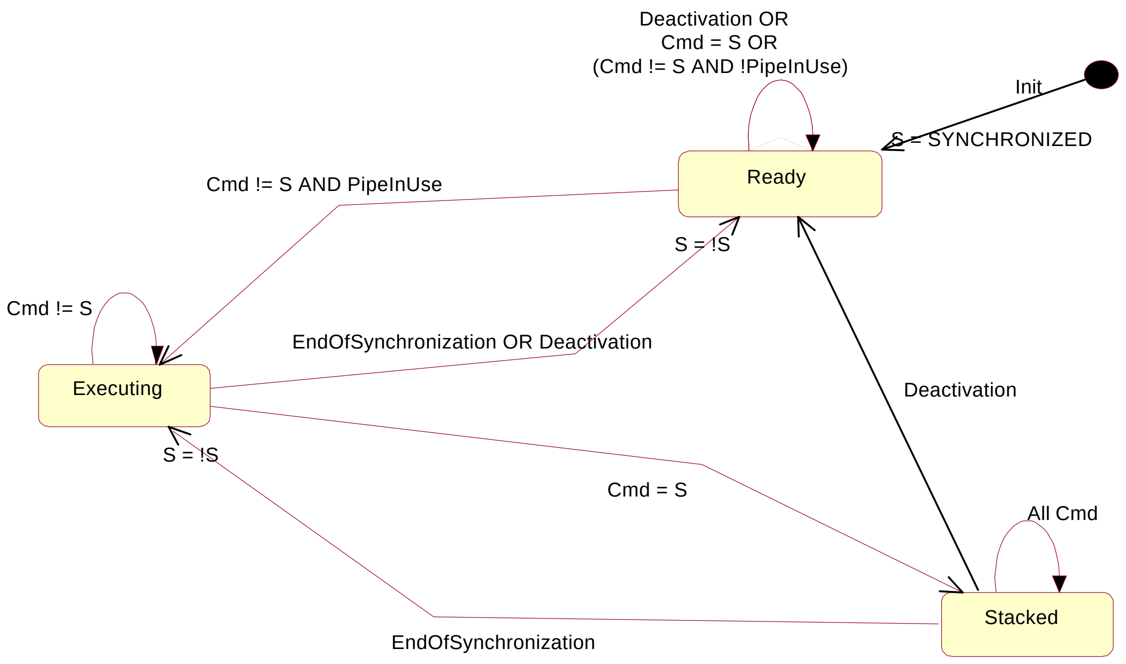
S: SYNCHRONIZED or UNSYNCHRONIZED. Note that there is a small difference between the value of this variable and the one returned by the SYN_SYNCHRONIZED status bit :

S remains SYNCHRONIZED until the slave is completely stopped.
SYN_SYNCHRONIZED is FALSE as soon as the generator starts to drive the slave.

PipeInUse: Indicates if the synchronizer is a part of a running pipe.

Commands

Cmd: start (SYNCHRONIZED) or stop (UNSYNCHRONIZED)
Deactivation: Deactivation of the running pipe.



Initialisation

Initially, the synchronizer is in the READY state and SYNCHRONIZED.

Behaviour when deactivated (not part of a running pipe)

In this mode, the synchronizer is always in the READY state and transitions from SYNCHRONIZED to UNSYNCHRONIZED are done instantaneously. Note that in case of a stop command, the specified rest position is stored.

Behaviour when part of an active pipe

In this mode, transitions from SYNCHRONIZED to UNSYNCHRONIZED are executed using the type of curve specified in the declaration.
Suppose the synchronizer in the READY state and SYNCHRONIZED. If a stop command is triggered, the synchronizer goes to the EXECUTING state and the ramp generator drives the output until the master reaches $m_{e.}$ The synchronizer then returns to the READY state and S is UNSYNCHRONIZED.

What happens if a new command is asked for before the previous one was completed (ready condition true).

- A command interrupting itself (A start interrupting a start or a stop interrupting a stop) has no effect. The second command is simply ignored.
- A command interrupting a contradictory one is stacked and will be treated after the current one has completed its job if the job was already in progress. Note that only two successive commands can be stacked. Once in the STACKED state all new commands are ignored.

Stacked commands

Stacking commands can be very useful to phase out immediately after having terminated a phase in (or conversely) without having to take particular care to the exact (probably narrow) dead zone between the commands.
Suppose the synchronizer in the EXECUTING state and a contradictory command being triggered, the synchronizer goes to the STACKED state and the ramp generator continues to drive the output until the master reaches m_e of the first command. The synchronizer then returns to the EXECUTING state to execute the second command. The ramp generator drives the output until the master reaches m_e of the second command and the synchronizer finally returns to the READY state.

Deactivation

When a pipe containing a synchronizer pipe block is deactivated, the synchronizer goes directly to the READY state and the S variable is set to the same value it would get at the end of the execution of the current commands if there were no deactivation. For example, if the synchronizer is in the EXECUTING state, reacting to a stop command, then S will be set to UNSYNCHRONIZED. On an other hand, if the synchronizer is in the STACKED state, reacting to a stop-start command, then S will be set to SYNCHRONIZED.

Periodicity

The relations between rendezvous and periodicity is to be explained:

- Both input and output periodicity are identical. They are inherited from the preceding block.
- If a command is given too late, the pipe block waits for the next possible occurrence of the starting condition (generally one period later).
- If Δs is larger than one period, the behaviour is the same as if no periodicity was specified and then the periodicity was applied at the very end of the block.

Modification of the phasing value

Each command uses the current value of OUTPUT_PHASING. If this field changes its value during the process, the value stored at the beginning is kept. If a new command is triggered then the new value is taken in account (only by the new command)

Change of direction

A subtle point is to detail what happens when the input changes its direction while phasing in or out.
If, after having began to phase in, the input stops and goes back over the starting point, the output follows down the same curve and then stops at the previous rest position. Note that the synchronizer stays in the EXECUTING state. For the synchronizer to leave this state, the input would have to change its direction again and reach the m_e value.

July 31, 1998 Yannis Jeannotat

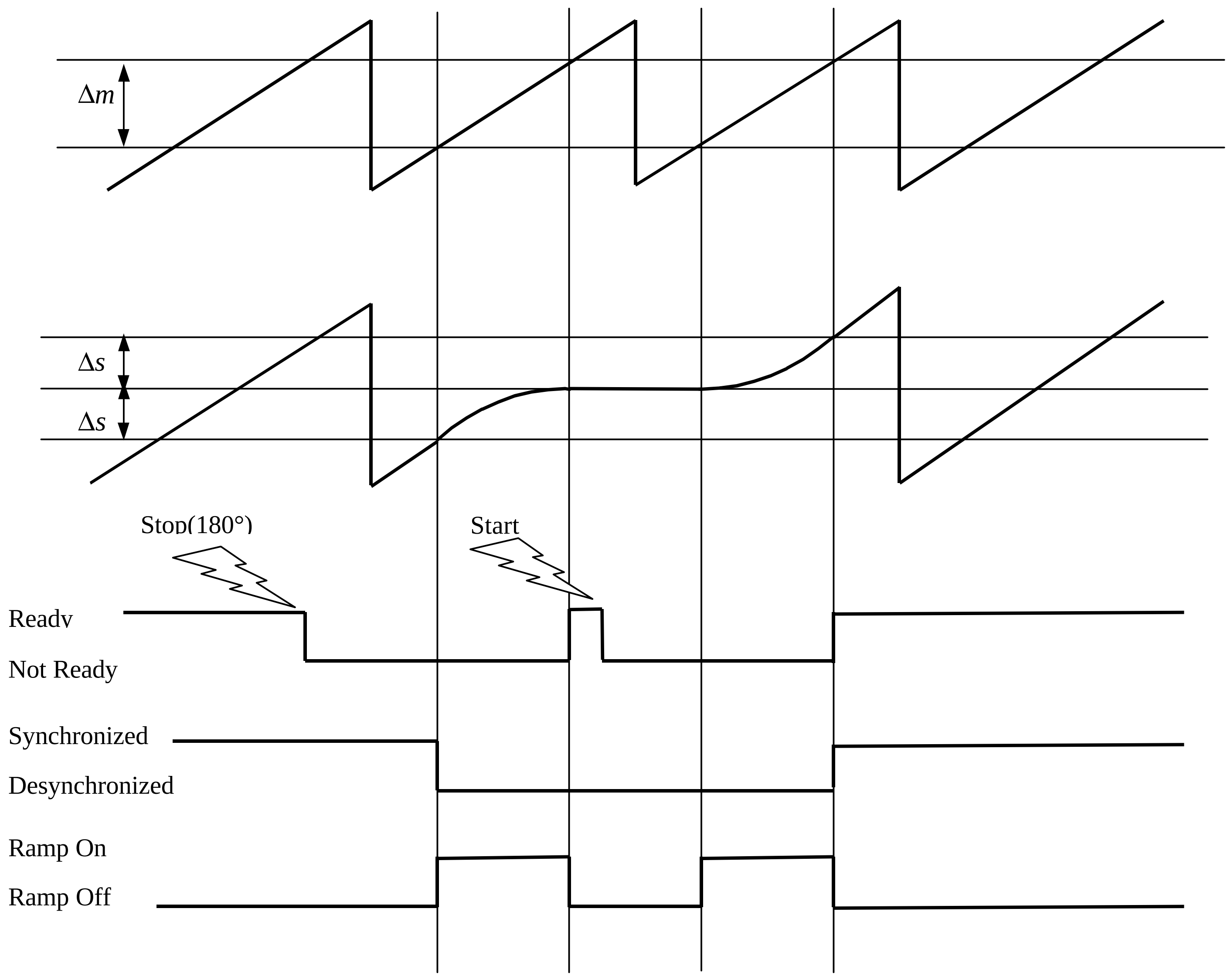
Example 1

Curve Type = Parabola
$\Delta s = 90°$

Using the above table :

Stop :
$\Delta m = 2\,\Delta s = 180°$
Master position at the beginning = rest_position – Δs = 180 – 90 = 90°
Master position at the end = rest_position + Δs = 180 + 90 = 270°

Start :
$\Delta m = 2\,\Delta s = 180°$
Master position at the beginning = rest_position – Δs = 180 – 90 = 90°
Master position at the end = rest_position + Δs = 180 + 90 = 270°



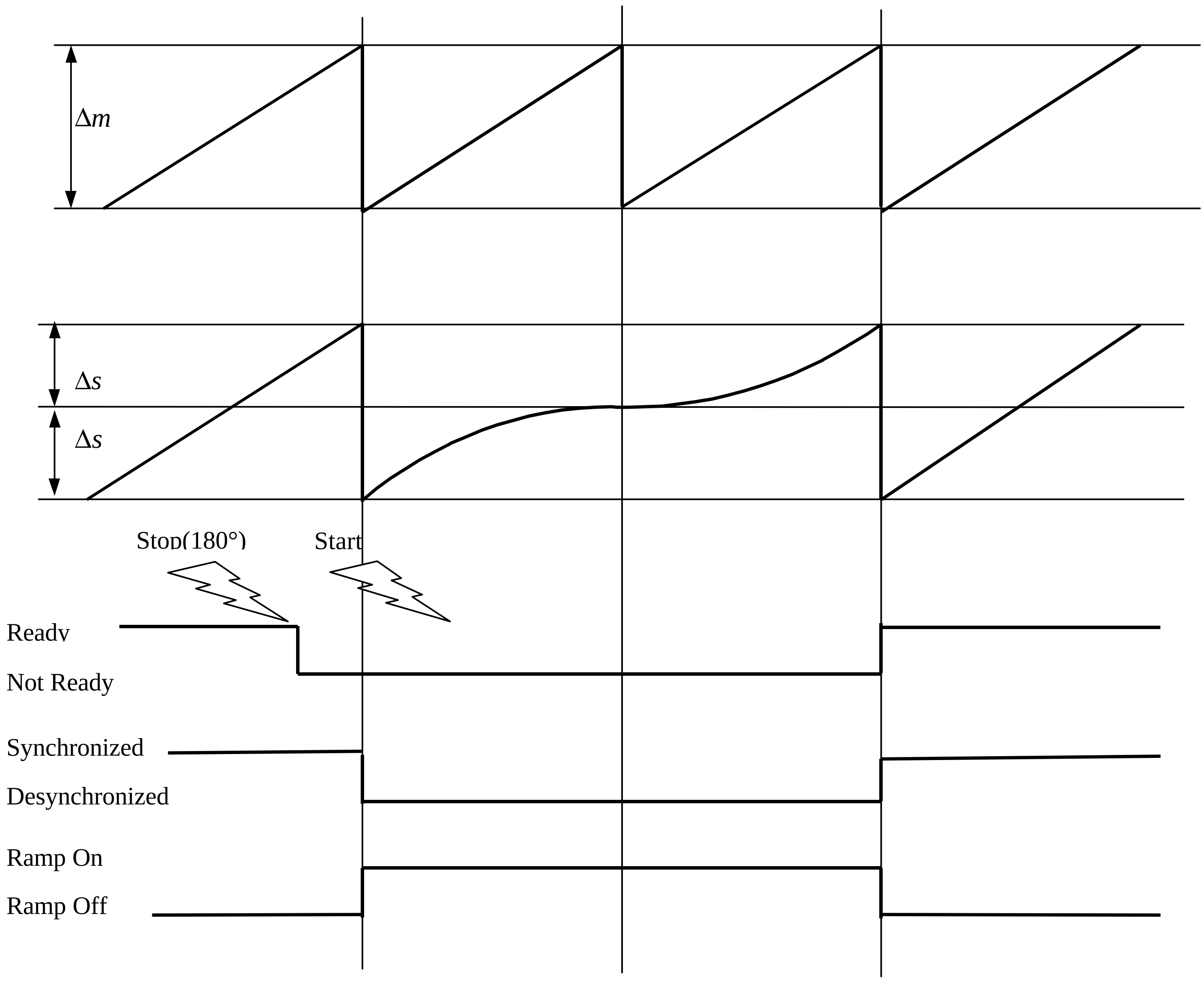
Example 2

Curve Type = Parabola
$\Delta s = 180°$

Using the above table :

Stop :
$\Delta m = 2\Delta s = 360°$
Master position at the beginning = rest_position – Δs = 180 – 180 = 0°
Master position at the end = rest_position + Δs = 180 + 180 = 360°

Start :
$\Delta m = 2\Delta s = 360°$
Master position at the beginning = rest_position – Δs = 180 – 180 = 0°
Master position at the end = rest_position + Δs = 180 + 180 = 360°


Δm
Δs
Δs
Stop(180°)
Start
Ready
Not Ready
Synchronized
Desynchronized
Ramp On
Ramp Off