ROOT Logic for the Global Control of apeNEXT

June 20, 2006

H. Simma

1 Introduction

Like all APE100 and APEmille, also apeNEXT has a hierarchical tree network of interrupt and control signals for the global control of the machine. In the following these signals[1] will simply be called "root signals" and the devices which handle them "root logic".

The hierarchy levels of the apeNEXT root logic correspond to the different sub-systems of the machine:

- Top (N Crates = $4N \times 8 \times 8$ Nodes) or[2]
- Rack (2 Crates = $8 \times 8 \times 8$ Nodes)
- Crate (4 Units = $4 \times 8 \times 8$ Nodes)
- Unit (4 Boards = $4 \times 2 \times 8$ Nodes)
- Board (2 Halfboards = $4 \times 2 \times 2$ Nodes)
- Halfboard (8 Nodes = $2 \times 2 \times 2$ Nodes = "Cube")
- Node

The root signals interconnect the different root levels in a tree topology (see Fig. 1) and allow to transfer information or commands from a lower to a higher level ("upward") or vice versa ("downward"). The purpose of the root logic is to

- collect and reduce the upward signals/commands
- generate and distribute the downward signals/commands

[1] We will not strictly distinguish between the "signals" at the electrical level and the logical "commands" which are handled by the root logic through such signals

[2] The Rack-level is skipped if a Top-level with $N > 2$ is present

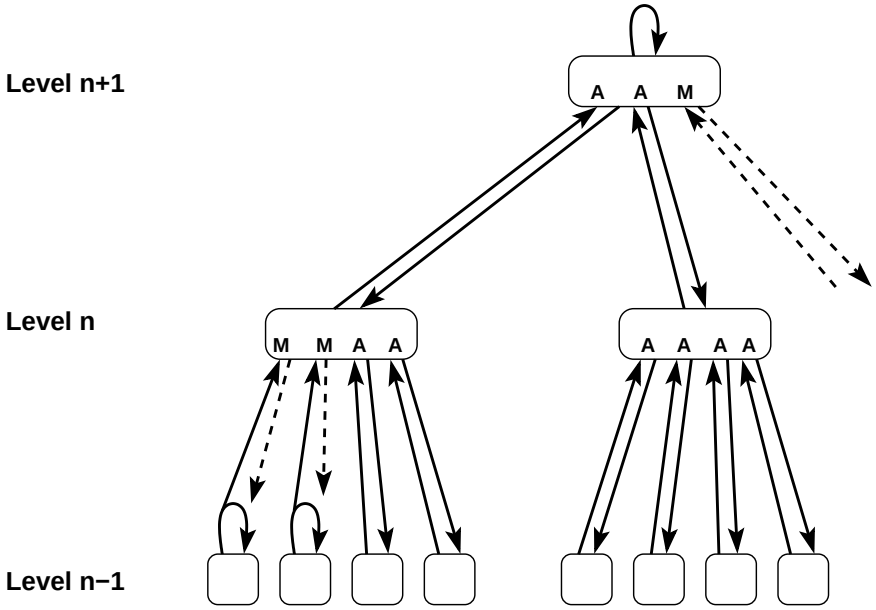

Figure 1: Example of the interconnection between three different root levels: At the lowest level, two sub-systems are closed and their upward signals are masked on the next higher level. The masking of incoming upward signals is indicated by an "M". The highest level is closed and the additional sub-system(s) of this level, which are not shown explicitly in the figure, are masked.

The signals generated after the reduction of the incoming upward signals are always propagated to the next higher root level.

At each root level the incoming *upward* signals from the individual subsystems which correspond to the next lower level can be masked by configuration registers ("partition registers").

For the generation and propagation of *downward* signals each root level can be configured as "closed" or "open". In *closed* mode, the downward signals are generated according to the result of the reduction of the incoming upward signals (which is always propagated upward), while all incoming downward signals are ignored. Therefore, this level of the root logic operates as the top level for all its sub-systems.

In *open* mode the root logic simply forwards the incoming downward signals

to the next lower level (which in turn might ingore them if it is closed itself).

Upward commands are:

- Kill (**KILL**)
- All (**ALL**)
- Trigger (**TRIG**)
- True local condition on all lower levels/nodes (**TRUE**)
- False local condition on any lower level/node (**FALSE**)

These commands are encoded and decoded in such a way that the commands in the above list have decreasing precedence from the first to the last.

The **KILL** command is issued whenever an (unmasked) exception occurs on a node (identified at the lowest, i.e. halfboard, level of the root logic by the `STATUS_1` line from a nodes being high).

The **ALL** command indicates to the next higher root level that all nodes of the sub-sustem are in I^2C-mode (identified at the lowest level of the root logic by the `STATUS_2` line from all nodes being high).

At the lowest level of the root logic, the generation of the upward **TRIG** command may be configured in two different ways: either if one (or more) of the nodes is in I^2C-mode (default configuration) or if any of the nodes has executed the `TXTRIGGER` instruction[3] . In the first case, an upward **TRIG** signal at any level of the root logic indicates that *any* node of the corresponding sub-system (but not all of them) is in I^2C-mode.

The two commands **TRUE** and **FALSE** correspond to the boolean AND of the local condition values on each node. Note that the upward (and subsequent downward) propagation of these two commands implies also a synchronisation of the nodes: the root logic first waits until each lower level is transmitting its condition value and only then evaluates and propagates the boolean AND over all the received condition values[4].

[3]Currently in the configuration registers of `PALREG` the bit to selected between the two sources of the **TRIG** signal is not yet implemented, and an upward **TRIG** signal is only generated if any of the nodes is in I^2C-mode.

[4]The root logic actually works with a condition strobe **CSTR** and the condition value **CVAL**. Therefore, transmission of just the two local condition commands **TRUE** and **FALSE** requires at least two upward signal wires.

Downward commands are:

- Reset (**RST**)
- Kill (**KILL**)
- Trigger (**TRIG**)
- True global condition (**TRUE**)
- False global condition (**FALSE**)

Also these commands are encoded and decoded in such a way that the commands in the above list have decreasing precedence from the first to the last.

A downward **KILL** command is generated whenever an upward **KILL** command due to an exception on some node(s) reaches the highest relevant (i.e. closed) level of the root tree. In addition, a downward **KILL** command can be generated by writing the value 1 into a corresponding bit of the command registers (see `RCREG`, `U*CREG`, `CCREG` and `TCREG` below).

The generation of the downward **TRIG** command depends on the configuration of the root logic: It is either generated by writing the value 1 into a corresponding bit of the command registers (see `U*CREG`, `CCREG` and `TCREG` below) or upon receiving an upward **TRIG** command from a lower root level. The selection between these two behaviours can be configured (see `U*PREG`, `CPREG` and `TPREG` below).

2 Hardware Implementation

2.1 Encoding of upward commands

Command	Origin	Reduction	Signals 0 1 2
KILL	Node	any	1 1 0
ALL	Board	all	0 0 1
TRIG	Node	any	1 0 1
TRUE	Node	all	0 1 1
FALSE	Node	all	0 1 0
NOP	—	—	0 0 0

The duration of the upwards commands depends on their generation by all or any of the nodes. In fact, the nodes continue to generate the upward command until they receive a corresponding downward command from the root logic (see section 7.3 for further details).

2.2 Encoding of downward commands

Command	Origin	Duration	Signals IFS IFD1 IFD2
RST	Root	8 T_{RCLK}	dedicated wire
KILL	Node	any(N)	0 1 1
	Root	8 T_{RCLK}	
TRIG	Node	any(N)	0 1 0
	Root	8 T_{RCLK}	
TRUE	Node/Root	all(N)	0 0 0
FALSE	Node/Root	all(N)	0 0 1
NOP	—	—	1 0 0

The third column in the table specifies the duration of the command. The commands generated by the root logic itself have a duration of 8 clock cycles T_{RCLK} of the root logic. This duration is necessary to guarantee that the command is longer than

- the stability period (3 cycles of the clock `RCLK` of the root logic) requirement for the detection of downward commands in the lowest root level (see section **??** for further details)
- one period of the I^2C clock (to allow safe passing of signals into the domain of the I^2C clock)

2.3 Firmware of the root logic

The logic for Halfboard- and Board-level is contained in the FPGA (U21, also called PALREG) on each Processing Board ("BP") and will be called "ROOT0" in the following.

The logic for Unit- and Crate-level is contained in FPGA1 (U14) on the Root Board ("RB") and will be called "ROOT1"in the following. ROOT1 is connected with ROOT0 via the backplane.

The logic for Rack- and Top-level is in FPGA2 (U16) on the RB and will be called "ROOT2" in the following. ROOT2 can be connected with ROOT1 (of the same or of a different RB) via a pair of cables or by an internal bus (if on the same RB).

2.4 Configuration switches

The front-panel of the RB has two 4-bit DIP switches. A switch being in left position corresponds to the value 0. The switches are used as follows

Upper Switch			
Nr.	Color	VHDL	Description
1	brown	RT_ADD	RB Address
2	red	RT_ADD	RB Address
3	orange	RT_ADD	RB Address
4	yellow	RT_ADD	RB Address
Lower Switch			
1	brown	INT_CLK_EN_SW	Clock Selection
2	red	SW_INTERNAL	Interconnect Selection
3	orange	—	—
4	yellow	SW_FIXJ10	Signal swapping for J10

The RB Address has no relevance for the bahaviour of the root logic, but can be used to allow the SW to identify e.g. the Top-level RB in a multi-crate configuration.

The switch for the Clock Selection determines whether the clock of the RB (which in turn is distributed over the backplane to all boards) is taken from an internal or external source:

- When INT_CLK_EN is 0, the "Clk In" connector on the front-panel must be connected to an external **PECL** clock (e.g. from connector "Clk Out" of an other RB)
- When INT_CLK_EN is 1, an "internal" clock source is used. Depending on the setting of the hardware jumper JPR888 this is the internal quartz oscillater (jumper setting 1–2) or an external **TTL** clock generator connected to "Tst Clk" on the front-panel.

The switch for the Interconnect Selection determines whether ROOT1 is internally connected to the ROOT2 (on the same RB) by the interconnect bus (SE_INTERNAL = 1), or whether the upward and downward signals of ROOT1 to and from higher root levels are routed to the connector pair "SRt" on the front-panel (SE_INTERNAL = 0)

2.5 Reset Signals

The following reset sources have to be distinguished

(RBot) Push-botton or powerup circuit of the RB

(RCmd) Command registers of the RB

(RI2C) I^2C-interface of the RB

(BBot) Push-botton or powerup circuit of the processing board (PB)

(BBP) Reset signal from the RB, distributed by the backplane to all PBs

(BI2C) I^2C-interface of the processing board

(BC) Command registers of processong board

On the other hand we distinguish the following (disjunct) reset domains which are reset by corresponding signals

- ROOT2 internal registers of the root logic (in RCL domain)
- ROOT2 configuration registers (in I^2C address space and SCL domain)
- ROOT1 internal registers of the root logic (in RCL domain)
- ROOT1 configuration registers (in I^2C address space and SCL domain)
- ROOT0 internal registers of the root logic (in RCL domain)
- ROOT0 and PAL configuration registers (in I^2C address space and SCL domain)
- xxx

3 Configuration Registers

All configuration registers to configure the various levels of the root logic are (only) accessible via I^2C.

These configuration registers and all other I^2C-registers can only be reset by the reset command over the I^2C channel (generated by writing 0x00400000 to the HIB command register), but they are **not** reset by a **RST** command issued at any level of the root logic.

The only exception from this rule is the `NCREG` register of the nodes (because at the node-level the **RST** command from the root logic performs a power-

up reset of the chip (which resets all internal registers, including the I^2C interface and all its registers, as well as the memory interface[5]).

The philosophy of not resetting the I^2C registers by the **RST** command from the root logic was adopted to avoid screwing up of the I^2C channels. This might happen because the information on the last value written to IDREG, which the HIB internally keeps to optimise repeated accesses to the same device, may becomes inconsistent. The natural cure of this inconsistency (at least when the reset is generated by SW) would be that the operating system enforces the writing of the IDREG in the first I^2C access after a reset through the root logic. Unfortunately, the HIB currently does not correctly support the command flag to force writing of the IDREG before an I^2C access (rather the HIB completely blocks the I^2C channel if this flag is activated!).

On the other hand, we have found situations (e.g. after a temporary change of the SCL frequency on the HIB) that the I^2C interface on the PB may become blocked in such a way that it does not receive or execute any more a reset command issued via the I^2C channel. Therefore, it might be desirable to change the above reset handling such that a **RST** from the root logic also resets the I^2C registers (to avoid the necessity to reset them by pushing the reset bottins on each individual PB of the blocked channel!).

3.1 Node-level

At the node-level there is only the 1-bit configuration register `NCREG` (at I^2C address 0x3) which closees the root signals of each individual node when set to 1.

3.2 Board-level (ROOT0)

At board-level the root logic in ROOT0 is configured only through the register `RCREG` (currently at I^2C address 0x2). The bits 16:31 represent the mask for the individual nodes. The bits 32:34 close the ROOT0 logic for half-board 0

[5]Therefore, after any **RST** command from the root logic, the startup sequence of the memory interface has to be performed by writing the command 0x5 into the I^2C register CREG of the nodes.

and 1 and for the full board, respectively, when set to 1. By setting bit 36 to 1 a **RST** command is generated for all nodes of the board (and bit 37 is raised when the command is completed).

As the layout of the configuration register(s) for the ROOT0 logic might change in the future, the relevant reference for detailed information is the file `$NXTPROJECT/doc/hubert/I2CREGS`.

3.3 Unit- and Crate-level (ROOT1)

For further details we refer to the file `$NXTPROJECT/doc/hubert/ROOTREGS`

3.4 Top-level (ROOT2)

For further details we refer to the file `$NXTPROJECT/doc/hubert/ROOTREGS`

4 System Configuration

To avoid an unnecessary increase of the latency of the root logic, no Rack-level logic is used in systems with more than one rack. This is achieved by direct cable connections from all (but one) Crate-level RBs to a unique Top-level RB (which we always assume here to be the RB of Crate 0). We call this configuration in the following "Multi-Crate Configuration".

In addition, it may become desirable that the RBs can be configured and connected in a "Multi-Rack Configuration" which allows to partition the system as a Top-level system and/or several Rack-level systems without re-cabling. Currently this possibility is not yet implemented.

Each RB must have ROOT1 and ROOT2 firmware loaded (otherwise the I^2C signals are not propagated to connector and the entire I^2C channel is inaccessible). This constraint should be removed in a future release of the firmware by activating the I^2C signals to ROOT2 only if a corresponding configuration switch is set (e.g. by using a separate switch or coupled to `INT_CLK_EN_SW`).

In the following description of the cabling and hardware configuration of the root logic we refer to the layout and labling of the front panel of the RB as shown in Figure 2.

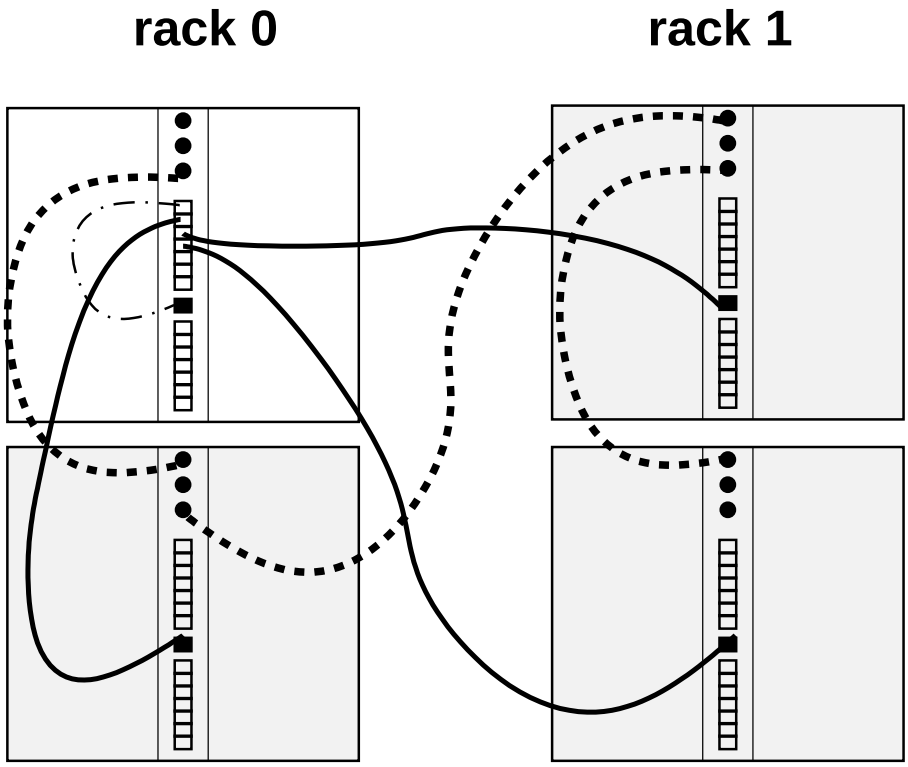

Figure 2: Cabling of the RBs of 4 crates in Multi-Crate configuration with the master RB (white) in the upper crate of rack 0. The RBs of all other crates are slaves (grey), i.e. sub-systems of the ROOT2 in the master RB. Dotted lines show the cables of the daisy chain for the clock distribution. Solid lines represent pairs of CAT5 cables for the root signals. The dash-dotted line is the optional cable for the external connection between ROOT1 and ROOT2 of the master RB.

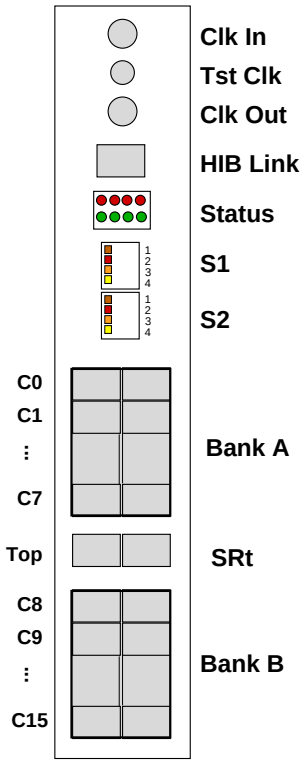

Figure 3: Front panel of the RB with the positions and labeling of connectors, configuration switches, etc.

4.1 Clock distribution

Master: External TTL clock input (`INT_CLK_EN` = 1, i.e. brown switch of lower group in right position)

Master: PECL output on JC15 (lowest of the three clock connectors)

Slaves: External PECL clock input (`INT_CLK_EN` = 0, i.e. brown switch of lower group in left position) to JC1 (=external clock cable into upper-most of the three clock connectors)

4.2 Cabling of root signals in Multi-Crate configuration

The RB of each crate in a Multi-Crate configuration must be connected to the RB of the "master" crate by a pair of CAT5 cables.

The connection between the top-level ROOT2 and the ROOT1 of the master crate itself (which is on the same RB) can be done by a pair of cables analog to the ones used for the other crates, or by the internal interconnect bus (by setting a configuration switch, see 4.3 below).

4.3 Switch setting in Multi-Crate configuration

4.3.1 Upper group S1

The switches of the upper group S1 are irrelevant for the functionality of the ROOT hardware. They may be (but currently are not) used by the OS, e.g. to identify or check the hardware ID of the crates.

4.3.2 Lower group S2

The first (brown) switch of the lower group S2 selects the clock input (see also 4.1):

- Left position (off) requires an external (differential) clock signal to be feed to the connector "Clk In".
- Right position (on) uses (depending on the settings of jumper JPR888) the internal clock generator (if jumper setting is 1-2) or an exteral TTL clock signal to be feed to the connector "Tst Clk" (if jumper setting is 2-3)

The second (red) switch of the lower group S2 selects whether the cable pair (if present) or the internal interconnect bus is to be used to connect ROOT2 and ROOT1 on the master RB (see 4.2).

- Left position (off) requires and uses the external cable pair between

(usually) the upper-most connectors of Bank A and the connectors SRt of the same RB.

- Right position (on) uses the internal interconnect bus.

  In multi-crate configuration, this setting can only be used on the master RB (but not on a slave RB which always must use the external cables to communicate with ROOT2)!

The third (orange) switch of the lower group S2 is currently unused.

The last (yellow) switch of the lower group S2 selects the negation of the upward signals from ROOT1 to pin B05/B06 at J10 (SRt). This selection is supported starting with firmware version 3 (see bits 15 downto 8 of FSREG). To allow any connection of ROOT1 to ROOT2 by external cables, this switch must be in

- Left position (off) for RBs starting from revision C (where the connections to pin B05/B06 are fixed and no swapping is needed)
- Right position (on) for old RBs with revision A (where the connections to pin B05/B06 are incorrect and explicit swapping by ROOT1 is needed)

4.4 Cabling and switch settings in Multi-Rack configuration

To be defined (not yet implemented)

5 Firmware Revisions

The type, version, and revision of the actual firmware can checked by reading the corresponding Firmware Status Regiser (`FSREG`)

ROOT0 (in PALREG)

- Revision 05: Compiled with out-of-date[6] sources of ROOT0

[6] with incorrect generation of downward **TRIG**on Halfboard 0: `TDOWN_H(0) <= '0' when ALL_S2_H(1) = '0'`

- Revision 05.1 (26.4.2005): Re-compiled with up-to-date sources

The revision number read from `FSREG` of PALREG is not yet following a clear and documented logic ... ask Sergio.

ROOT1 (Type = 01)

- Version 02 Revision 00: First experimental version
- Version 02 Revision 01 (4.5.2005): `INT_CLK_EN` connected to `INT_CLK_EN_SW` (instead of fixed value 1)
- Version 02 Revision 02 (10.6.2005): Fixed incorrect address for `RADCREG`
- Version 03 Revision 00 (22.5.2006): Added support to optionally swap signals B05/B06 to J10 by configuration switch (instead of fixed swapping) and changed reset value of RREQ to 0

ROOT2 (Type = 02)

- Version 02 Revision 00: First experimental version
- Version 02 Revision 01 (4.5.2005): Fixed bug in handling of `TSEL`
- Version 03 Revision 00 (22.5.2006): Added support to optionally swap signals B05/B06 to J10 by configuration switch (insstead of fixed swapping)

The *version* number of ROOT1 and ROOT2 should always be equal (i.e. it is supposed to be incremented whenever the singnalling between the two devices changes).

6 Missing Features and Improvements

PALREG and HIB:

- Fix of command flag to enforce writing of IDREG (HIB)
- Improved I^2C-protocol (uni-directional use of reset wire) (PALREG+HIB)
- Implement parity check in I^2C-protocol (PALREG+HIB)
- Simplified implementation of HSM (PALREG)

- New LED layout (PALREG)
- Fix format of `FSREG` (PALREG)
- Replace baroque test-logic for tests with mini-backplane by a configuration register which simply selects the encoding of upward signals (PALREG)
- Implementation of missing configurability (ROOT0)
- Implement KILL request by `RCREG` (ROOT0)
- Implement TSEL (ROOT0)

ROOT1:

- Support absence of ROOT2 by configuration switch
- Drop `INT_CLK_EN` output to recover INTERC[10] on pin_55
- Fix reset of RDONE by botton reset of RB (need asyncronous reset in pulse.vhd?)
- Check/fix ADC
- Revise LED layout (green = not any(I2C)?)

ROOT2:

- Implement support for Multi-Rack configuration
- ...

See also file root8/2do.

7 Implementation Details

7.1 I^2C interface

Currently the RB uses a simplified I^2C interface, which does not

- take into account only the `SDA` line of the I^2C signal (but not the second wire originally foreseen for reset) when determining the start of an I^2C transaction

- perform a dummy cycle of the FSM when a device is accessed, which is not on the RB
- take into account the 4 `RT_ADD` switches to assign individual addresses to different RBs

Therefore, currently each RB must be on a individual I^2C channel.

7.2 Names used in VHDL sources and schematics

The following table shows the relation between the signal and component names indicated on the front panal of the RB and the actual names used in the schematics and VHDL sources.

Front Panel	VHDL/Schematics
Connectors	
Clk In	JC1 , `CLK_IN`
Clk Out	JC15, `CLK_OUT`
Tst Clk	JC13, `CLK_TEST`
SRt	J10
Bank A	J11 . . . J14
Bank B	J15 . . . J18

7.3 Signal handshake between nodes and root logic

The node generates and decodes the upward and downward root signals according to the following table

Command	IFS	IFD2	IFD1
hline **KILL**	0	1	1
TRIG	0	0	1
TRUE	0	0	0
FALSE	0	1	0
NOP	1	X	X

Upward commands:

Due to an unfortunate feature of the `WHERE` on the node, the generation of the upward **KILL** command is not guaranteed in certain situations (e.g. while the node transmits a **TRUE** or **FALSE** command, i.e. during a "global if").

Therefore, the ROOT0 logic ignores upward **KILL** commands from the node via the root signals (i.e. `IFS`, `IFD2`, `IFD1`) from the node and instead generates upward **KILL** commands depending on the value of the `STATUS_1` pin of the node (1 in case of exceptions, 0 otherwise).

Moreover, by default the ROOT0 logic ignores upward **TRIG** commands from the node and instead generates upward **TRIG** and **ALL** commands depending on the values of the `STATUS_2` pin of the node (1 in case of I^2C-mode, 0 otherwise).

Downward commands:

The ROOT0 logic generates the downward commands **KILL TRIG TRUE** and **FALSE** by activating the 3 root signals (i.e. `IFS`, `IFD2`, `IFD1`) towards the nodes. The signals towards the nodes are activated only when the incoming downward commands are stable for a certain number of N_{stab} clock cycles (see `gfilter.vhd`).

The nodes respond to the receipt of a downward command by deactivating the generation of the corresponding upward command (if any). This response arrives at the ROOT0 logic after the upward and downward propagation time through the entire root tree. To avoid that this propagation delay causes a large scheduling constraint on the distance between microcode of subsequent global IF operations (which would also depend on the depths of the root-tree), the true commands **TRUE** and **FALSE** are generated by the ROOT0 logic only for a fixed number of cycles. Any further incoming downward **TRUE** or **FALSE** commands are ignored by the ROOT0 logic (i.e. not propagated downward to the nodes) until such commands are absent for at least N_{stab} clock cycles (see `glock.vhd`), i.e. until the response of the node propagated through the entire root tree in a stable way. In this way, the minimal microcode distance between subsequent global IF operations is only limited to the number of processor clock cycles which correslonds to a time

little more[7] than $N_{stab} \times T_{RCLK}$

The **RST** command is performed by activating the dedicated wire `RESET_7512` which is directly connected to the reset pin of all nodes of the PB.

[7] because of sampling ambiguities by signals passing from the processor clock domain to the clock domain of the root logic and vice versa