> **Info (not part of baseline!)**
>
> This is an updated state diagram for Type 3 and Type 4 single-signature PDs. It addresses a number of issues. These changes are not compatible with Type 1 and Type 2, this baseline assumes that we will split the SD into legacy and new Types state diagrams.
>
> 1. The PD state diagram is stuck in IDLE due to the $((V_{PD} < V_{Reset}) +$!power_received) condition being True whenever !power_received. This is addressed by removing power_received from the entry conditions into IDLE and providing a different path back from the powered states. See yse-boodt_7_0116_idlestuck.pdf for the problem definition.
>
> 2. The PD state machine does not handle starting up with power_received = True. The expected behaviour is that it ends up in MDI_POWER1. This is handled by removing power_received from the entry into IDLE and the startup variable.
>
> 3. The calculation of pd_max_power is completely wrong. This has been fixed by changing the numeric values of pse_power_level to values from which the maximum power can be calculated. Also class_sig is renamed to pd_req_class.
>
> 4. The SD does not handle the case when voltage drops back into class or mark range when it is in the powered states. The original SD would have reset the PD to IDLE, making it forget any events. This is addressed by providing an MDI_NOPOWER state which traps the PD into an undefined condition until $V_{PD} < V_{Reset}$.
>
> 5. The PD is not required but allowed to show a valid detection signature in the IDLE state. On the first run when present_det_sig is not set yet this is OK. If however we return to IDLE from any other state, present_det_sig has the value FALSE, which requires the PD to "A non-valid PD detection signature is to be applied to the link".

33.3.3.2 Constants

Delete variable class_sig and insert pd_req_class in it's place:
pd_req_class

> A constant indicating the requested Class of the PD.
> Values:
>> 1: The PD requests Class 1.
>> 2: The PD requests Class 2.
>> 3: The PD requests Class 3.
>> 4: The PD requests Class 4.
>> 5: The PD requests Class 5.
>> 6: The PD requests Class 6.
>> 7: The PD requests Class 7.
>> 8: The PD requests Class 8.

33.3.3.3 Variables

Remove the pd_multi_event variable from the variable list.

> **Info (not part of baseline!)**
>
> The values of pse_power_level are changed to match with class numbers. This greatly simplifies the calculation of pd_max_power.

Change the pse_power_level variable as follows:

pse_power_level

A control variable that indicates to the PD the level of power the PSE ~~is supplying~~ can supply.

Values:

~~1~~ 3: The PSE has allocated the PD's requested power or Class 3 power, whichever is less (default).

~~2~~ 4: The PSE has allocated the PD's requested power or Class 4 power, whichever is less.

~~3~~ 6: The PSE has allocated the PD's requested power or Class 6 power, whichever is less.

~~4~~ 8: The PSE has allocated the PD's requested power or Class 8 power, whichever is less.

> **Info (not part of baseline!)**
>
> The goal is to make the behaviour of the PD undefined when it has first received power and then the PD voltage is brought down into the class/mark/detection range. A proper reset requires $V_{PD} < V_{Reset}$. This is done through the pd_undefined variable and the state MDI_NOPOWER.

Add pd_undefined as follows:

pd_undefined

A control variable that indicates that the PD is in an undefined condition. The PD may or may not show a valid or invalid detection signature, may or may not draw mark current, may or may not draw any class current, may or may not show MPS and may change the pse_power_level variable.

Values:

FALSE: The PD is in a defined condition (default)

TRUE: The PD is in an undefined condition

> **Info (not part of baseline!)**
>
> We need a way to set present_det_sig as either FALSE or TRUE depending on implementation when in the IDLE state. Initially present_det_sig is undefined, but once the PD has been in the DO_DETECTION state the value will be FALSE. That would imply the PD is required to show an invalid detection in the IDLE state. The solution is to add a third possible value for present_det_sig: 'EITHER'.

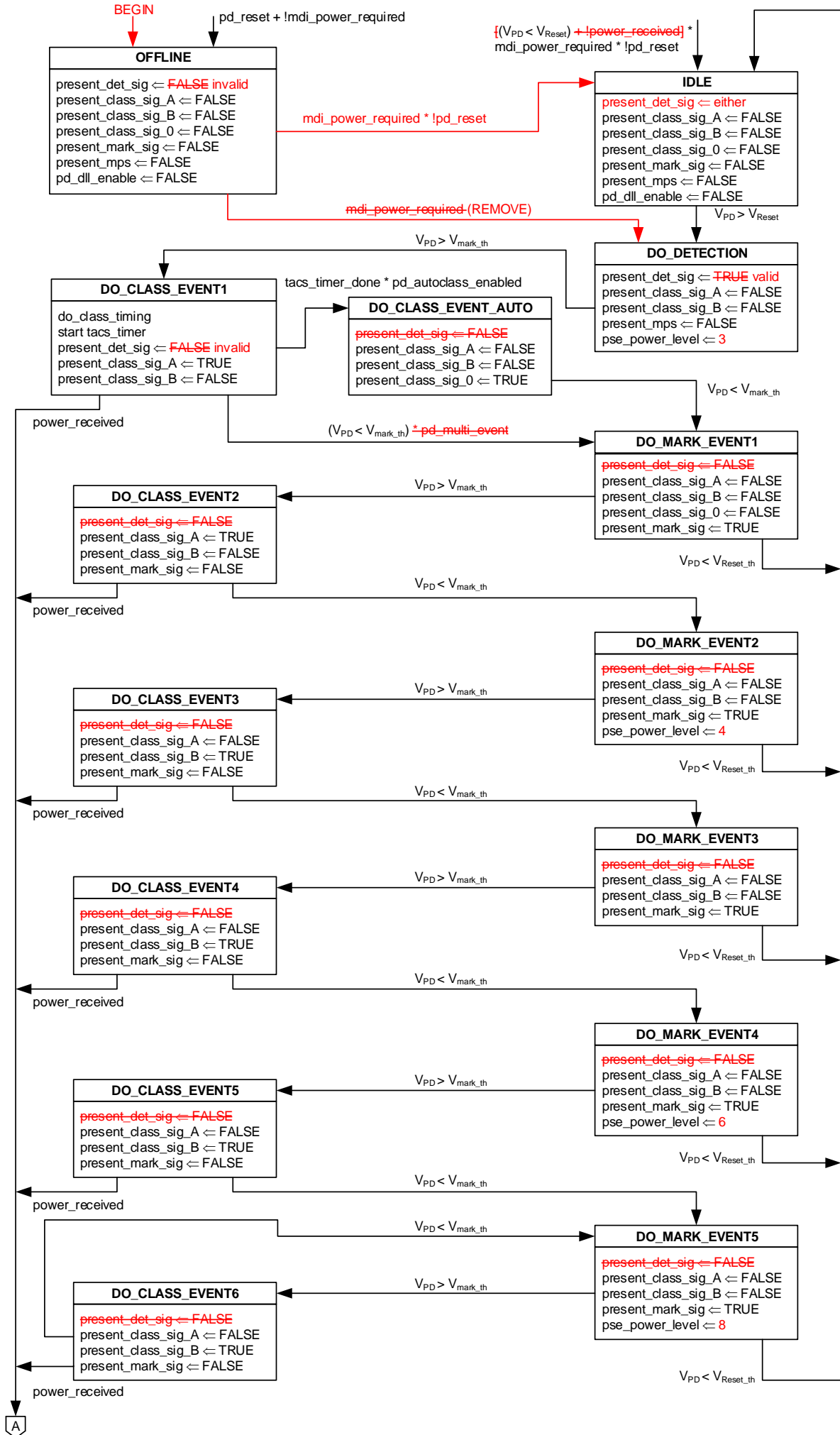### Change present_det_sig as follows:

present_det_sig

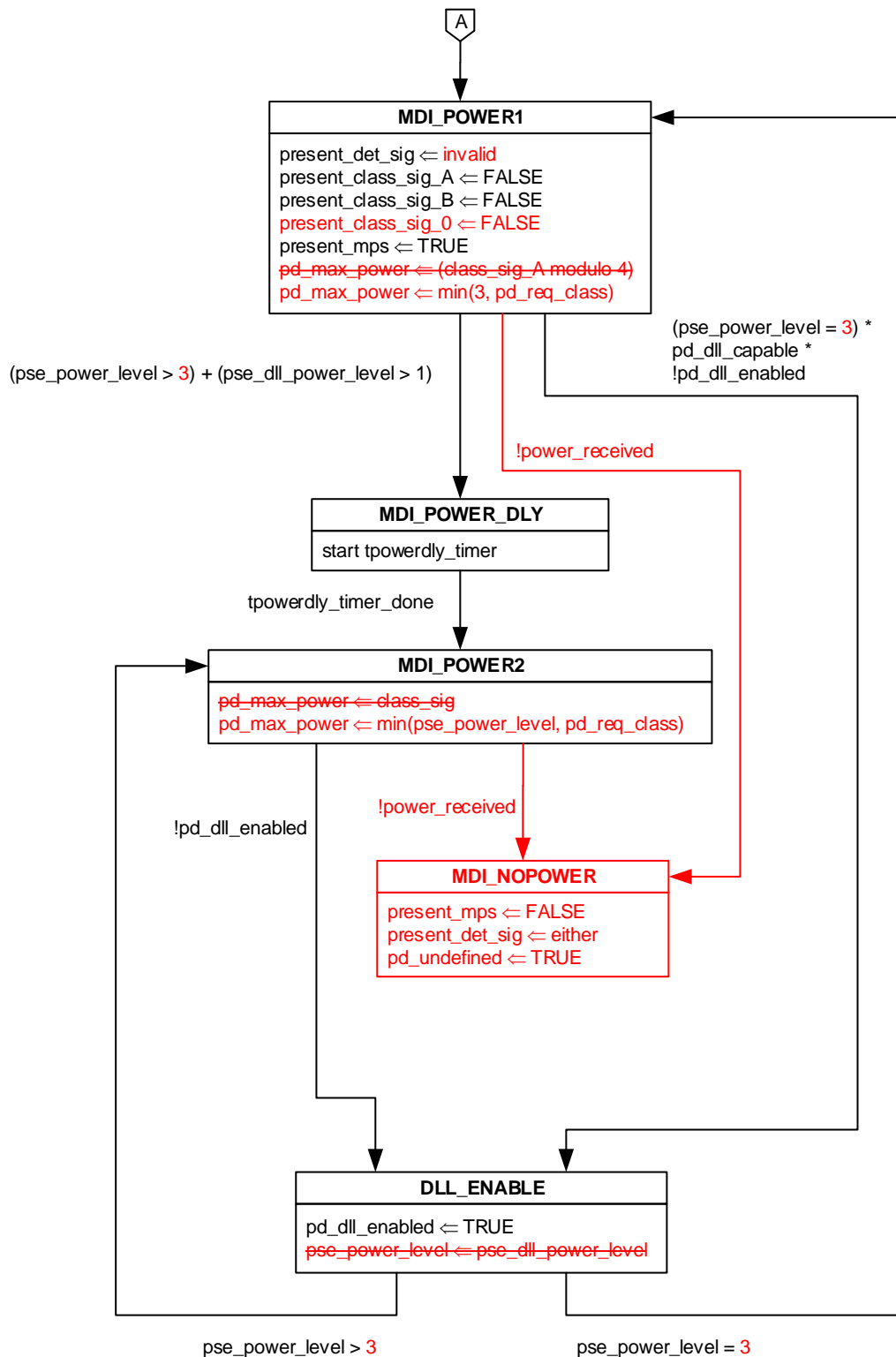Controls presenting the detection signature (see 33.3.4) by the PD.

Values:

~~FALSE~~ invalid: A non-valid PD detection signature is to be applied to the link.

~~TRUE~~ valid: A valid PD detection signature is to be applied to the link over each pairset.

either: Either a valid or non-valid PD detection signature may be applied to the link.

Change Figure 33–31 as follows (changes marked in red):

BEGIN

pd_reset + !mdi_power_required

{(V_PD < V_Reset) + !power_received] * mdi_power_required * !pd_reset

OFFLINE

present_det_sig ⇐ ~~FALSE~~ invalid
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ FALSE
present_class_sig_0 ⇐ FALSE
present_mark_sig ⇐ FALSE
present_mps ⇐ FALSE
pd_dll_enable ⇐ FALSE

mdi_power_required * !pd_reset

IDLE

present_det_sig ⇐ either
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ FALSE
present_class_sig_0 ⇐ FALSE
present_mark_sig ⇐ FALSE
present_mps ⇐ FALSE
pd_dll_enable ⇐ FALSE

V_PD > V_Reset

~~mdi_power_required~~ (REMOVE)

V_PD > V_mark_th

DO_DETECTION

present_det_sig ⇐ ~~TRUE~~ valid
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ FALSE
present_mps ⇐ FALSE
pse_power_level ⇐ 3

DO_CLASS_EVENT1

do_class_timing
start tacs_timer
present_det_sig ⇐ ~~FALSE~~ invalid
present_class_sig_A ⇐ TRUE
present_class_sig_B ⇐ FALSE

tacs_timer_done * pd_autoclass_enabled

DO_CLASS_EVENT_AUTO

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ FALSE
present_class_sig_0 ⇐ TRUE

V_PD < V_mark_th

power_received

(V_PD < V_mark_th) ~~* pd_multi_event~~

DO_MARK_EVENT1

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ FALSE
present_class_sig_0 ⇐ FALSE
present_mark_sig ⇐ TRUE

V_PD > V_mark_th

DO_CLASS_EVENT2

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ TRUE
present_class_sig_B ⇐ FALSE
present_mark_sig ⇐ FALSE

V_PD < V_Reset_th

V_PD < V_mark_th

power_received

DO_MARK_EVENT2

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ FALSE
present_mark_sig ⇐ TRUE
pse_power_level ⇐ 4

V_PD > V_mark_th

DO_CLASS_EVENT3

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ TRUE
present_mark_sig ⇐ FALSE

V_PD < V_Reset_th

V_PD < V_mark_th

power_received

DO_MARK_EVENT3

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ FALSE
present_mark_sig ⇐ TRUE

V_PD > V_mark_th

DO_CLASS_EVENT4

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ TRUE
present_mark_sig ⇐ FALSE

V_PD < V_Reset_th

V_PD < V_mark_th

power_received

DO_MARK_EVENT4

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ FALSE
present_mark_sig ⇐ TRUE
pse_power_level ⇐ 6

V_PD > V_mark_th

DO_CLASS_EVENT5

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ TRUE
present_mark_sig ⇐ FALSE

V_PD < V_Reset_th

V_PD < V_mark_th

power_received

V_PD < V_mark_th

DO_MARK_EVENT5

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ FALSE
present_mark_sig ⇐ TRUE
pse_power_level ⇐ 8

V_PD > V_mark_th

DO_CLASS_EVENT6

~~present_det_sig ⇐ FALSE~~
present_class_sig_A ⇐ FALSE
present_class_sig_B ⇐ TRUE
present_mark_sig ⇐ FALSE

power_received

V_PD < V_Reset_th

A

A

MDI_POWER1

present_det_sig ⟸ invalid
present_class_sig_A ⟸ FALSE
present_class_sig_B ⟸ FALSE
present_class_sig_0 ⟸ FALSE
present_mps ⟸ TRUE
~~pd_max_power ⟸ (class_sig_A modulo 4)~~
pd_max_power ⟸ min(3, pd_req_class)

(pse_power_level = 3) *
pd_dll_capable *
!pd_dll_enabled

(pse_power_level > 3) + (pse_dll_power_level > 1)

!power_received

MDI_POWER_DLY

start tpowerdly_timer

tpowerdly_timer_done

MDI_POWER2

~~pd_max_power ⟸ class_sig~~
pd_max_power ⟸ min(pse_power_level, pd_req_class)

!power_received

!pd_dll_enabled

MDI_NOPOWER

present_mps ⟸ FALSE
present_det_sig ⟸ either
pd_undefined ⟸ TRUE

DLL_ENABLE

pd_dll_enabled ⟸ TRUE
~~pse_power_level ⟸ pse_dll_power_level~~

pse_power_level > 3   pse_power_level = 3

33.3.6 PSE Type identification

The default value of pse_power_level is ~~1~~ 3. After a successful Multiple-Event Physical Layer classification has completed the pse_power_level is set to either ~~1, 2, 3, or 4~~ 3, 4, 6, or 8. After a successful Data Link Layer classification has completed, the pse_power_level is set to either 1, 2, 3 or 4.