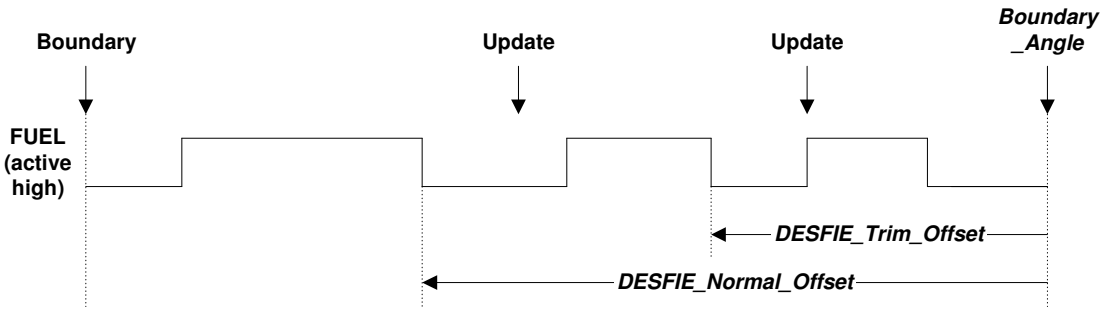
eTPU Function Implementation:

Dual End-Angle Sequential Fuel Injection Algorithm



MCD – 5417

Revision 2.0

March 17, 2009

DELPHI

CONFIDENTIAL

Microcode Technology
Powertrain Electronics Software Technology
M/S CT-40D, Delphi E & S, Kokomo, Indiana 46904-9005

This page intentionally left blank

Table of Contents

eTPU Function:
Dual End-Angle Sequential Fuel Injection Algorithm

1. Introduction

The Dual End-Angle Sequential Fuel Injection function for the eTPU (DESFIE) is designed to generate fuel pulses in either a simultaneous or sequential delivery mode. It may be assigned to any eTPU channel. The user may define the fuel output signal to be either active high or active low.

For each fuel channel, the host CPU must determine the amount of desired fuel, the fueling boundary angle, and the end-of-injection offsets that determine the timing and position of the fuel pulses to be delivered by the eTPU. A targeted trim fuel pulse will be delivered if the amount of desired fuel increases by more than a defined amount after the normal fuel pulse has ended. Further trim pulses (untargeted) may also be delivered as necessary if enabled by the host.

The eTPU will automatically initiate fuel delivery at each boundary, unless the host CPU has set the amount of desired fuel to 0. The host may update any of the fuel parameters at any time.

At the end of each fuel pulse, the eTPU will interrupt the host CPU and calculate actual delivered fuel for the given fuel channel. The eTPU will also calculate delivered fuel whenever the host issues an update or a boundary with a fuel pulse in progress. When the eTPU has finished executing a boundary, the host should immediately read the actual delivered fuel, then reset the value to 0.

2. System Overview

2.1 Inputs

There are no hardware inputs required for operation of this function.

2.2 Outputs

There is one output produced by the operation of this function (see **Figure 1**).

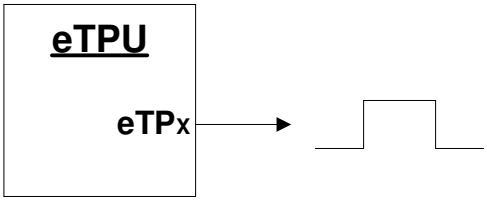


Figure 1 - Hardware Configuration

2.3 Simultaneous Fuel Delivery

The eTPU can deliver simultaneous fuel pulses on each channel, under control of the host CPU. The host must request a desired amount of fuel for each channel simultaneously. The simultaneous fuel pulses are start-time controlled, dependent upon when the host issues its requests for fuel. (See **Section 4.2.5** for more information.)

2.4 Sequential Fuel Delivery

The eTPU can deliver a sequential fuel pulse on each channel, under control of the host CPU. The host must request a desired amount of fuel for each channel separately, during the cylinder event for a given channel. The sequential fuel pulses are end-angle controlled, dependent upon the boundary angle and the end-of-injection offsets calculated by the host.

3. Memory Map

3.1 Parameter RAM

The parameter RAM map and bit definitions for the DESFIE function are shown in **Figure 2** and **Figure 3** respectively.

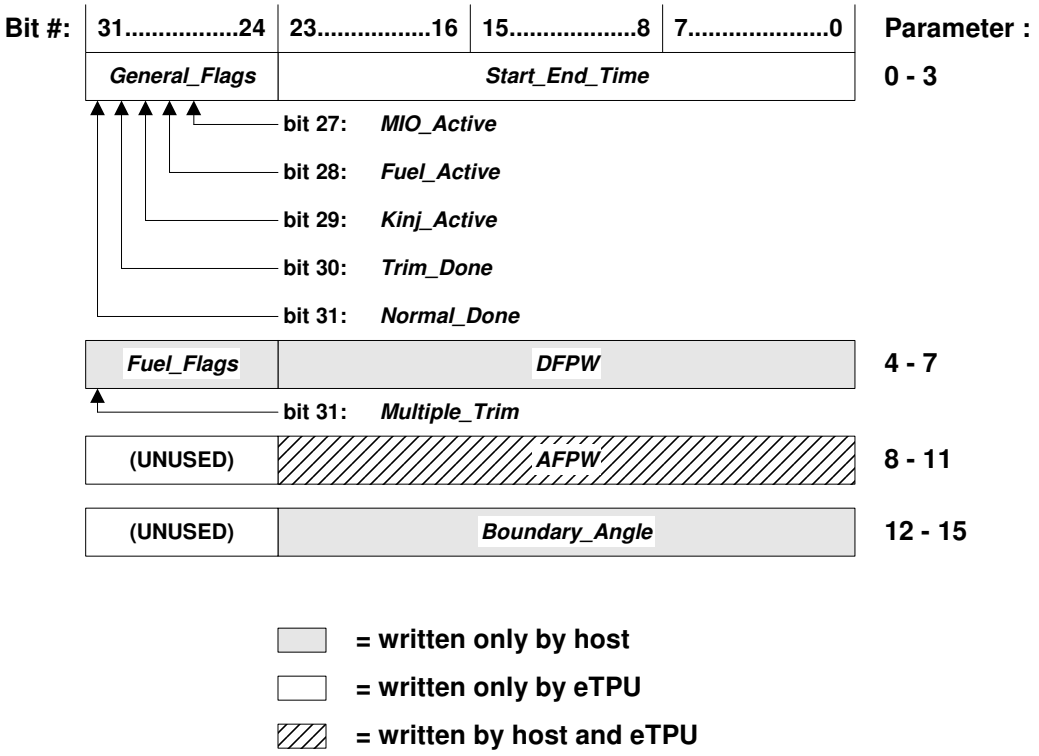


Figure 2 - Parameter RAM Map

Host Service Requests:

HSR7: Shutdown
HSR6: Boundary
HSR5: Initialize
HSR4: Update
HSR3: (UNUSED)
HSR2: (UNUSED)
HSR1: (UNUSED)

Entry Point Flags:

Flag1: (UNUSED)
Flag0: (UNUSED)

Function Mode Bits:

FM1: *Active_High*
0 = Output signal is active low
1 = Output signal is active high
FM0: *Use_TCR2*
0 = Use TCR1 time base
1 = Use TCR2 time base

***General_Flags*:**

Normal_Done
0 = Normal fuel pulse not done
1 = Normal fuel pulse done
Trim_Done
0 = Targeted trim fuel pulse not done
1 = Targeted trim fuel pulse done
Kinj_Active
0 = *Kinj* not in progress
1 = *Kinj* in progress
Fuel_Active
0 = Fuel delivery not in progress
1 = Fuel delivery in progress
MIO_Active
0 = *Min_Inj_Off* not in progress
1 = *Min_Inj_Off* in progress

***Fuel_Flags*:**

Multiple_Trim
0 = Multiple trim fuel pulse(s) not enabled
1 = Multiple trim fuel pulse(s) enabled

Figure 3 - Bit Definitions

3.2 Global RAM

The global RAM parameters accessed by the DESFIE function are shown in **Figure 4**. See the Application Implementation document for the actual memory locations of these parameters.

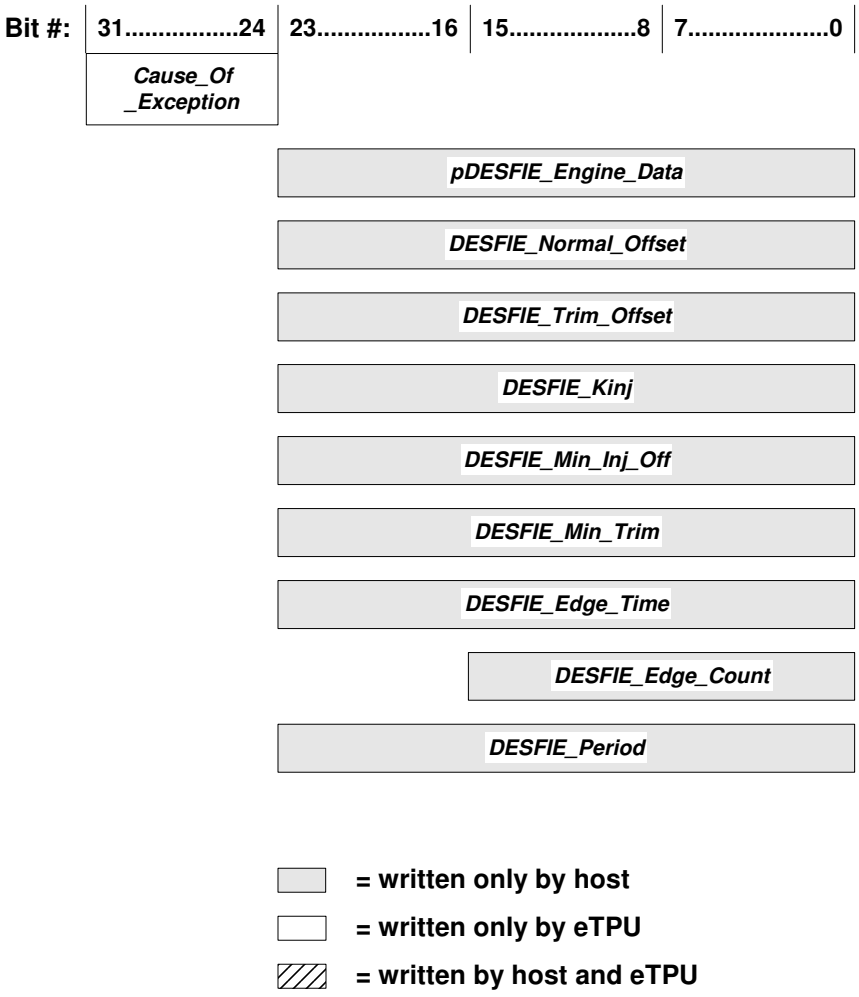


Figure 4 - Global RAM for DESFIE

3.3 Parameter RAM Definitions

The parameter RAM definitions are summarized in **Table 1**.

Parameter	Range	Units	CPU Access	eTPU Access
Local Parameter RAM				
General_Flags:				
- ***Normal_Done***	0 – 1	Flag	-	R/W
- ***Trim_Done***	0 – 1	Flag	-	R/W
- ***Kinj_Active***	0 – 1	Flag	-	R/W
- ***Fuel_Active***	0 – 1	Flag	-	R/W
- ***MIO_Active***	0 – 1	Flag	-	R/W
Start_End_Time	0 – 0xFFFFFF	TCRx Units	-	R/W
Fuel_Flags:				
- ***Multiple_Trim***	0 – 1	Flag	W	R
DFPW	0 – 0x7C0000	TCRx Units	W	R
AFPW	0 – 0x7C0000	TCRx Units	R/W	R/W
Boundary_Angle	0 – 0xFFFFFF	Low- or Med-Res Counts * 256	W	R
(FM1) ***Active_High***	0 – 1	Flag	W	R
(FM0) ***Use_TCR2***	0 – 1	Flag	W	R
Global RAM				
Cause_Of_Exception	0 – 0xFF	ETPU Channel #	R	W
pDESFIE_Engine_Data	0 – 0xFFFFFF	eTPU Address	W	R
DESFIE_Normal_Offset	0 – 0xFFFFFF	Low- or Med-Res Counts * 256	W	R
DESFIE_Trim_Offset	0 – 0xFFFFFF	Low- or Med-Res Counts * 256	W	R
DESFIE_Kinj	0xFE0000 – 0x01FFFF	TCRx Units	W	R
DESFIE_Min_Inj_Off	0 – 0x01FFFF	TCRx Units	W	R

DESFIE_Min_Trim	0 – 0x01FFFF*	TCRx Units	W	R
DESFIE_Edge_Time	0 – 0xFFFFFF	TCRx Units	W	R
DESFIE_Edge_Count	0 – 0xFFFF	Low- or Med-Res Counts	W	R
DESFIE_Period	0 – 0xFFFFFF	TCRx Units	W	R

* To disable all trim fuel pulses, set ***DESFIE_Min_Trim*** = 0xFFFFFF.

Table 1 - Parameter RAM Definitions

3.3.1 Host CPU Write

The following parameters are written only by the host CPU:

Fuel_Flags **Bit 31**:

Multiple_Trim **Bit 31** - Set (1) to enable multiple fuel trim pulses; cleared (0) to disable multiple fuel trim pulses (ie, only one targeted trim pulse will be allowed).

DFPW **Bits 23-0** - Desired amount of fuel for a given fuel channel's cylinder event; the amount of fuel that the eTPU will attempt to deliver. Units determined by ***Use_TCR2***.

Boundary_Angle **Bits 23-0** - The angle at which the next fuel **Boundary HSR** will be issued to the eTPU. Units are low- or med-res counts * 256.

pDESFIE_Engine_Data **(global) Bits 23-0** – Pointer to global engine position data used by DESFIE function (***DESFIE_Edge_Time***, ***DESFIE_Edge_Count*** and ***DESFIE_Period***).

NOTE: ***pDESFIE_Engine_Data*** **must be set equal to the address of *DESFIE_Edge_Time*.**

DESFIE_Normal_Offset **(global) Bits 23-0** - The desired offset angle between the end of the normal fuel pulse and the next ***Boundary_Angle***. Units are low- or med-res counts * 256.

DESFIE_Trim_Offset **(global) Bits 23-0** - The desired offset angle between the end of the first (targeted) trim fuel pulse and the next ***Boundary_Angle***. Units are low- or med-res counts * 256.

DESFIE_Kinj **(global) Bits 23-0** - Injector on/off time correction factor; added to each separate fuel pulse to insure that the correct amount of fuel is delivered. ***DESFIE_Kinj*** **may be a negative value!** Units determined by ***Use_TCR2***.

DESFIE_Min_Inj_Off **(global) Bits 23-0** - Minimum injector off time between any two fuel pulses (can be violated at a fuel boundary under certain conditions - see **Section 4.1.2**). Units determined by ***Use_TCR2***.

DESFIE_Min_Trim **(global) Bits 23-0** - Minimum trim fuel pulse width (minus 1 timer count) that the eTPU will deliver, not including ***DESFIE_Kinj*** time. Units determined by ***Use_TCR2***.

NOTE: To make *DESFIE_Min_Trim* inactive, the host should set it = 0x000000. To disable <u>all</u> trim fuel pulses, the host should set *DESFIE_Min_Trim* = 0xFFFFFF.

DESFIE_Edge_Time **(global) Bits 23-0** – Edge time as determined by host CPU using low-or med-res data. Units determined by ***Use_TCR2***.

DESFIE_Edge_Count **(global) Bits 15-0** – Edge count as determined by host CPU using low- or med-res data.

NOTE: The eTPU will read *DESFIE_Edge_Time* and *DESFIE_Edge_Count* coherently, and the host must also write them coherently!

DESFIE_Period **(global) Bits 23-0** – Period as determined by host CPU using low- or med-res data. Units determined by ***Use_TCR2***.

3.1.1.1 Function Mode Bits

The two Function Mode (FM) bits are located in the eTPUCxSCR register. These bits are written only by the host CPU:

Active_High	**(FM1)** Set (1) when the DESFIE output signal is to be active high (ie, high = fuel on); cleared (0) when the DESFIE output signal is to be active low (ie, low = fuel on).
Use_TCR2	**(FM0)** Set (1) when TCR2 is to be used as the DESFIE time base; cleared (0) when TCR1 is to be used as the DESFIE time base. ***Use_TCR2*** determines units for ***Start_End_Time***, ***DFPW***, ***AFPW***, ***DESFIE_Kinj***, ***DESFIE_Min_Inj_Off***, ***DESFIE_Min_Trim***, ***DESFIE_Edge_Time*** and ***DESFIE_Period***.

3.3.2 eTPU Write

The following parameters are written only by the eTPU:

General_Flags	**Bits 31-28**:
Normal_Done	**Bit 31** - Set (1) when the normal fuel pulse ends; cleared (0) at the next **Boundary HSR**.
Trim_Done	**Bit 30** - Set (1) when the first (targeted) trim fuel pulse ends, or when more fuel is requested after EOIT2 with no trim pulse(s) yet delivered; cleared (0) at the next **Boundary HSR**.
Kinj_Active	**Bit 29** – Set (1) when ***DESFIE_Kinj*** is in progress (at the beginning of a fuel pulse); cleared (0) otherwise. ***Kinj_Active*** **will not be set at the start of a fuel pulse when** ***DESFIE_Kinj*** **<= 0.**
Fuel_Active	**Bit 28** – Set (1) when fuel delivery is in progress (ie, fuel pulse is on but ***DESFIE_Kinj*** = 0); cleared (0) otherwise.
MIO_Active	**Bit 27** – Set (1) when ***DESFIE_Min_Inj_Off*** is in progress (at the end of a fuel pulse); cleared (0) otherwise.

Start_End_Time **Bits 23-0** - Updated with the current TCRx value when a fuel pulse begins or ends; when ***DESFIE_Kinj*** is done; and when the host CPU issues an **Update HSR** with fuel on. Units determined by ***Use_TCR2***.

Cause_Of_Exception **(global) Bits 31-24** – DESFIE channel number, written when a global exception occurs.

3.3.3 eTPU Write / Host CPU Write

The following parameters are written by the eTPU and by the host CPU:

AFPW **Bits 23-0** - Actual amount of fuel delivered for a given fuel channel's cylinder event. For each individual fuel pulse, ***AFPW*** is calculated as fuel pulse width – ***DESFIE_Kinj***. (Note that if ***DESFIE_Kinj*** is a negative value, ***AFPW*** will be larger than the actual fuel pulse width!) Updated by the eTPU at the end of each fuel pulse; when the CPU issues a **Boundary HSR** with fuel on; and when the host CPU issues an **Update HSR** with fuel on. Units determined by ***Use_TCR2***.

NOTE: AFPW must be read and cleared to 0x000000 by the host as soon as the eTPU is finished processing the Boundary HSR.

3.4 Initialization

The host CPU should initialize each DESFIE channel as follows:

1. Write the encoded value for the DESFIE primitive to the channel's field within the correct channel function select register (CFSx). See the Application Implementation document.
2. Set ***Active_High*** (FM1) and ***Use_TCR2*** (FM0) to the desired states.
3. Issue an **Initialize Host Service Request** (%101).
4. Enable the DESFIE channel by assigning to it a non-zero priority (CPR > %00).

When an **Initialize HSR** is issued, the eTPU will respond by configuring the DESFIE channel, canceling any pending fuel event and shutting the fuel output signal off.

NOTE: If *Active_High* = 0 and no fuel pulse was in progress at the time of the HSR, a low glitch lasting approximately 2 microcycles will occur on the output pin (see the Application Implementation document to convert microcycles into real time based on crystal frequency).

4. Operation

A general overview of the operation of the DESFIE function is shown in **Figure 5**:

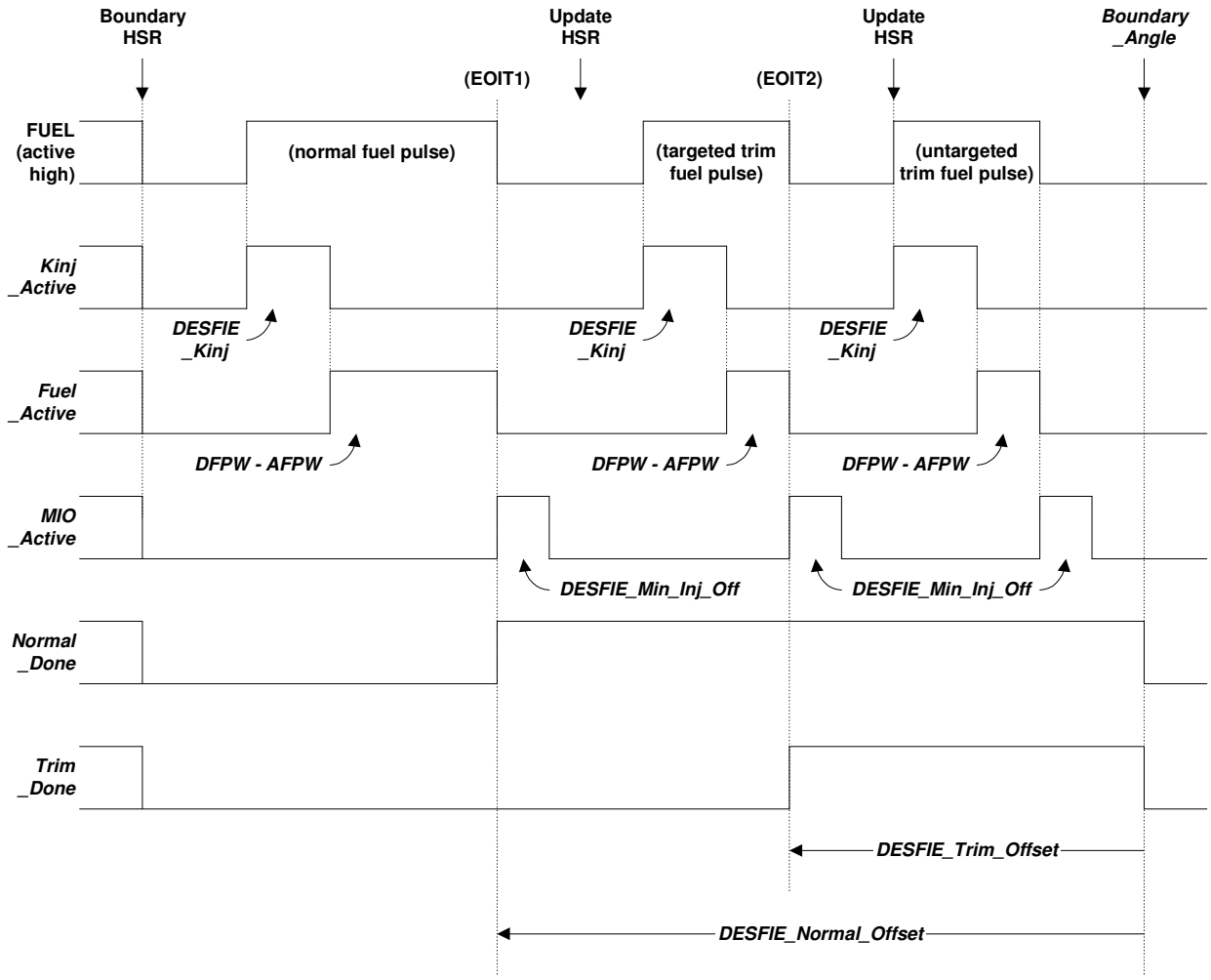


Figure 5 - General Operation

4.1 Boundary Host Service Request

To initiate targeted (sequential) fueling, the host CPU must issue a **Boundary HSR** (%110) to the eTPU. (See **Section 4.3.1** for more information on fuel boundaries.) Prior to doing this, the host must provide the following information to the eTPU: ***Boundary_Angle***, ***DESFIE_Normal_Offset***, ***DFPW***, ***DESFIE_Kinj***, and ***DESFIE_Min_Inj_Off***. (Note that ***Boundary_Angle*** must contain the **next** projected boundary angle, not the current one.)

When the eTPU receives a **Boundary HSR**, it will perform the following actions:

1. Turn off fuel (if a pulse is in progress), and cancel any fuel event that may be pending.

2. Clear ***Normal_Done*** and ***Trim_Done***.

3. Update ***AFPW*** (if a fuel pulse was in progress).

4. Attempt to set up the leading edge of the next normal fuel pulse following the boundary rules described in **Section 4.1.1**.

As soon as the eTPU has finished processing the **Boundary HSR**, the host CPU should read and store ***AFPW***, then reset ***AFPW*** to 0. **This is the only time (except for initialization) that the host should write to *AFPW*!** The eTPU will not use ***AFPW*** in its calculations until either the end of the normal fuel pulse, or the first **Update HSR** that occurs. The host should, therefore, have ample time to read and clear ***AFPW***.

NOTE: Under certain conditions – if *DFPW* = 0, or if (*DFPW* + *DESFIE_Kinj*) <= 0, or if the fuel pulse's leading edge is > 0x7FFFFF timer counts in the future – the eTPU will not initiate a fuel pulse. In such cases, it is up to the host to issue an Update HSR at a later time to begin fueling!

4.1.1 Boundary Rules

After ensuring that any fueling activity from the previous boundary cycle is shut off, the eTPU determines how to calculate the next normal fuel pulse per the following rules:

1. If no fueling activity was in progress at the boundary (***Kinj_Active*** = 0, ***Fuel_Active*** = 0, and ***MIO_Active*** = 0), then a normal fuel pulse will be targeted to end at the first end-of-injection target (EOIT1), an angular value calculated as ***Boundary_Angle – DESFIE_Normal_Offset***. EOIT1 is converted to a time value and used to calculate the leading edge of the normal fuel pulse, which will be set up to occur at (EOIT1 – ***DFPW*** – ***DESFIE_Kinj***).

2. If **Min_Inj_Off** was in progress at the boundary (***MIO_Active*** = 1) and (***DESFIE_Kinj*** + ***DFPW*** + ***DESFIE_Min_Inj_Off***) can be delivered before the next boundary is scheduled to occur, then **Min_Inj_Off** will be allowed to finish before attempting to calculate the leading edge of the normal fuel pulse.

3. If **Min_Inj_Off** was in progress at the boundary (***MIO_Active*** = 1) and (***DESFIE_Kinj*** + ***DFPW*** + ***DESFIE_Min_Inj_Off***) cannot be delivered

before the next boundary is scheduled to occur, then the leading edge of the normal fuel pulse will be set up to occur immediately (no targeting) and **Min_Inj_Off** will be truncated.

4. If a fuel pulse was in progress at the boundary (***Kinj_Active*** = 1 or ***Fuel_Active*** = 1) and (***DESFIE_Kinj*** + ***DFPW*** + ***DESFIE_Min_Inj_Off***) can be delivered before the next boundary is scheduled to occur, then the leading edge of the normal fuel pulse will be set up to occur **Min_Inj_Off** after fuel was shut off (no targeting).

5. If a fuel pulse was in progress at the boundary (***Kinj_Active*** = 1 or ***Fuel_Active*** = 1), and (***DESFIE_Kinj*** + ***DFPW*** + ***DESFIE_Min_Inj_Off***) cannot be delivered before the next boundary is scheduled to occur, but (***DESFIE_Kinj*** + ***DFPW***) can be delivered, then the leading edge of the normal fuel pulse will be set up to occur at (EOIT1 – ***DFPW*** – ***DESFIE_Kinj***).

6. If fuel delivery was in progress at the boundary (***Fuel_Active*** = 1) and (***DESFIE_Kinj*** + ***DFPW***) cannot be delivered before the next boundary is scheduled to occur, then the output fuel signal will be turned back on immediately and the trailing edge of the normal fuel pulse will be set up to occur ***DFPW*** after fuel was shut off (no targeting).

7. If **Kinj** was in progress at the boundary (***Kinj_Active*** = 1) and (***DESFIE_Kinj*** + ***DFPW***) cannot be delivered before the next boundary is scheduled to occur, then the output fuel signal will be turned back on immediately and **Kinj** will be allowed to finish before starting actual fuel delivery.

NOTE: In cases 6 and 7 (above), the output fuel signal will be shut off for up to 95 microcycles, but this is ignored by the fueling calculations. (See the Application Implementation document to convert microcycles into real time based on crystal frequency.)

4.2 Fuel Pulse Generation

4.2.1 Start of Fuel Pulse

Normally, the first portion of any fuel pulse is the injector offset (***DESFIE_Kinj***). At the leading edge, if ***DESFIE_Kinj*** > 0, the eTPU will set ***Kinj_Active***, store the

captured time in ***Start_End_Time***, and schedule a match to occur ***DESFIE_Kinj*** timer counts in the future.

However, if ***DESFIE_Kinj*** <= 0, this entire stage of the fuel pulse will be skipped. In this case, when a fuel pulse begins, ***Fuel_Active*** will be set immediately, while ***Start_End_Time*** will be set to a point ***DESFIE_Kinj*** before the start of the pulse (see **Figure 6**).

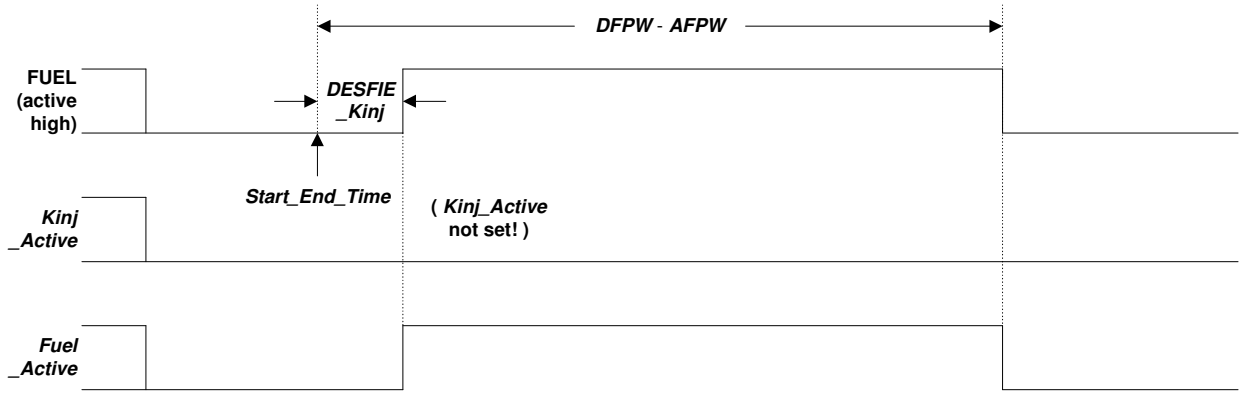


Figure 6 - Negative Injector Offset

4.2.2 Fuel Delivery

When the ***DESFIE_Kinj*** time expires, the eTPU clears the ***Kinj_Active*** flag, sets ***Fuel_Active***, and stores the match time in ***Start_End_Time***. For a normal fuel pulse, the trailing edge is set up to occur at ***Start_End_Time*** + ***DFPW***; for a trim fuel pulse, the trailing edge is set up to occur at ***Start_End_Time*** + (***DFPW*** – ***AFPW***).

4.2.3 End of Fuel Pulse

When any fuel pulse ends, the eTPU will issue an interrupt to the host CPU, update ***Start_End_Time***, clear ***Fuel_Active***, set ***MIO_Active***, update ***AFPW***, and schedule a match to occur ***DESFIE_Min_Inj_Off*** timer counts after the trailing edge. For a normal fuel pulse, ***Normal_Done*** will be set; for a targeted trim fuel pulse, ***Trim_Done*** will be set.

4.2.4 End of Minimum Injector Off-Time

When the ***DESFIE_Min_Inj_Off*** time expires, the eTPU clears ***MIO_Active***, then decides whether or not more fuel needs to be delivered during the current cylinder event. If (***DFPW*** - ***AFPW***) <= ***DESFIE_Min_Trim***, or ***DESFIE_Min_Trim*** = 0xFFFFFF (trim pulses disabled), or EOIT2 (***Boundary_Angle*** – ***DESFIE_Trim_Offset***) is too far in the future, then no further fuel will be delivered until either a **Boundary HSR** or an **Update HSR** intervenes. If more fuel **is** to be delivered, the eTPU decides whether the trim pulse is to be targeted (***Trim_Done*** = 0) or untargeted (***Trim_Done*** = 1). If it is to be a targeted trim pulse, then the leading edge will be scheduled to occur at (EOIT2 - (***DFPW*** - ***AFPW***) – ***DESFIE_Kinj***); if it is to be an untargeted trim pulse (and ***Multiple_Trim*** = 1), then the leading edge will be scheduled to start immediately.

Trim fuel pulses will be generated according to the procedures described in **Sections** Error! Reference source not found. **- 4.2.4**, with the following exceptions:

1. The amount of fuel to be delivered will always be (***DFPW*** - ***AFPW***), rather than simply ***DFPW***.
2. Targeting calculations will use ***DESFIE_Trim_Offset***, rather than ***DESFIE_Normal_Offset***.

4.3 Fuel Delivery Modes

4.3.1 Definition of Boundary

A boundary is the point at which fuel will be delivered to the next cylinder event for a given fuel channel. It is used to prevent fuel requested for a cylinder's fuel delivery cycle from crossing over into the next delivery cycle. The boundary is intended to lie just before an intake valve closing, but can be positioned anywhere within a low- or med-res interval and calculated by the host CPU.

4.3.2 Targeted Normal Fuel Pulse

If it is desired to issue only normal fuel pulses without any trim pulses, then ***DESFIE_Min_Trim*** should be set = 0xFFFFFF. This is a special case that will ensure that only one targeted normal fuel pulse will be issued per cylinder event, regardless of changes in ***DFPW*** (see **Figure 7**).

The host CPU calculates the end-of-injection target (EOIT1) as ***Boundary_Angle*** – ***DESFIE_Normal_Offset***. The eTPU calculates when the normal fuel pulse must start in order to meet this target (see **Section 4.1**). Note that the end-of-injection target will be violated if necessary to deliver all of the fuel that has been requested.

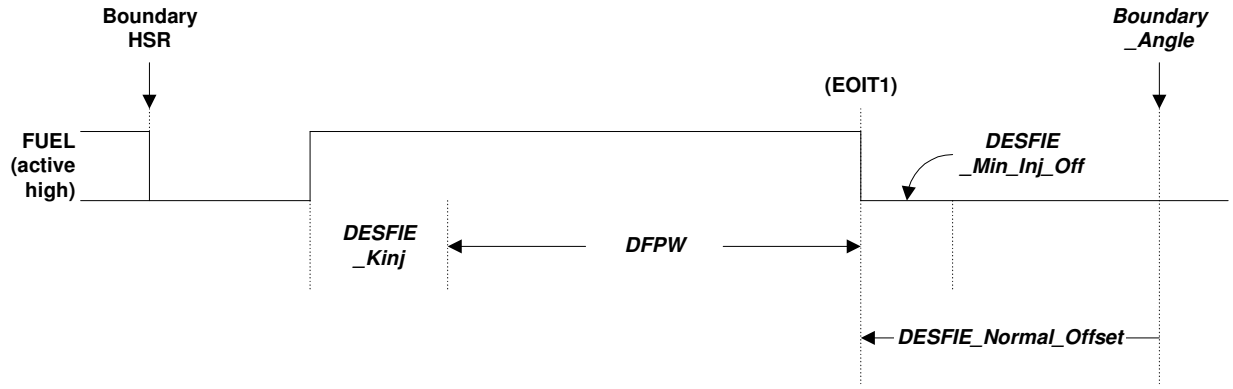


Figure 7 - Targeted Normal Fuel Pulse

4.3.3 Targeted Trim Fuel Pulse

If, after a normal fuel pulse has been delivered, the host CPU issues an **Update HSR** that increases the value of ***DFPW*** such that (***DFPW*** - ***AFPW***) > ***DESFIE_Min_Trim***, then the eTPU will issue a targeted trim pulse using (***Boundary_Angle*** – ***DESFIE_Trim_Offset***) as its end-of-injection target (EOIT2 - see **Figure 8**). The eTPU calculates the required start time for the targeted trim pulse in exactly the same way as for the normal pulse, except that it uses ***DESFIE_Trim_Offset*** instead of ***DESFIE_Normal_Offset***, and (***DFPW*** - ***AFPW***) instead of ***DFPW***. The procedures for generating the targeted trim pulse are also the same. Note that the end-of-injection target will be violated if necessary to deliver all of the fuel that has been requested.

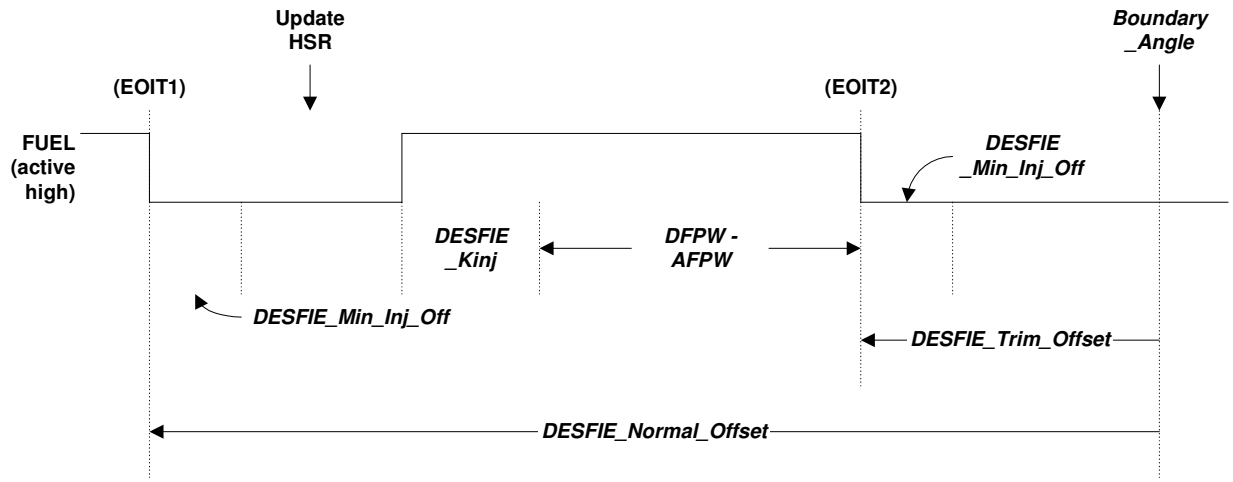


Figure 8 - Targeted Trim Fuel Pulse

The only exception to the above procedure is the case in which, with *Multiple_Trim* = 0, the host CPU does not issue an **Update HSR** until **after** EOIT2 (see **Figure 9**). In this case, even though the *Trim_Done* flag will be set, the eTPU **will not issue a trim pulse** during the current injection cycle. If *Multiple_Trim* = 1 in this situation, then the eTPU will deliver an untargeted trim pulse (see **Section 4.3.4**).

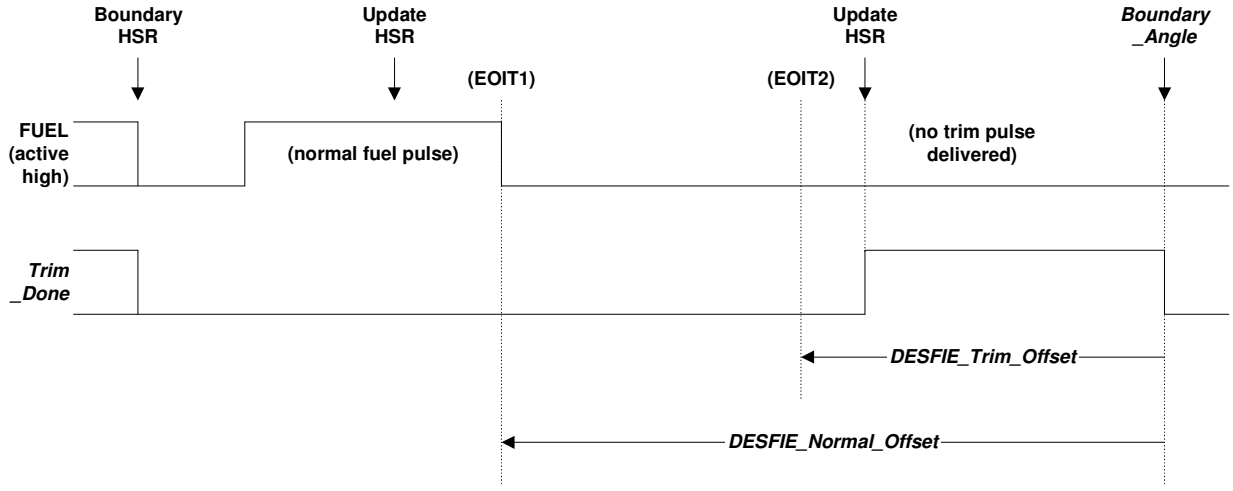


Figure 9 - Requesting Trim Pulse After EOIT2

4.3.4 Untargeted Trim Fuel Pulse(s)

If, when ***Trim_Done*** = 1, the host CPU issues an **Update HSR** which increases the value of ***DFPW*** such that (***DFPW*** - ***AFPW***) > ***DESFIE_Min_Trim***, and ***Multiple_Trim*** = 1, then the eTPU will immediately issue an untargeted trim pulse (see **Figure 10**). The procedures for generating an untargeted trim pulse are the same as for a targeted trim pulse (apart from the targeting calculations). There is no theoretical limit to the number of untargeted trim pulses that may be generated in this manner.

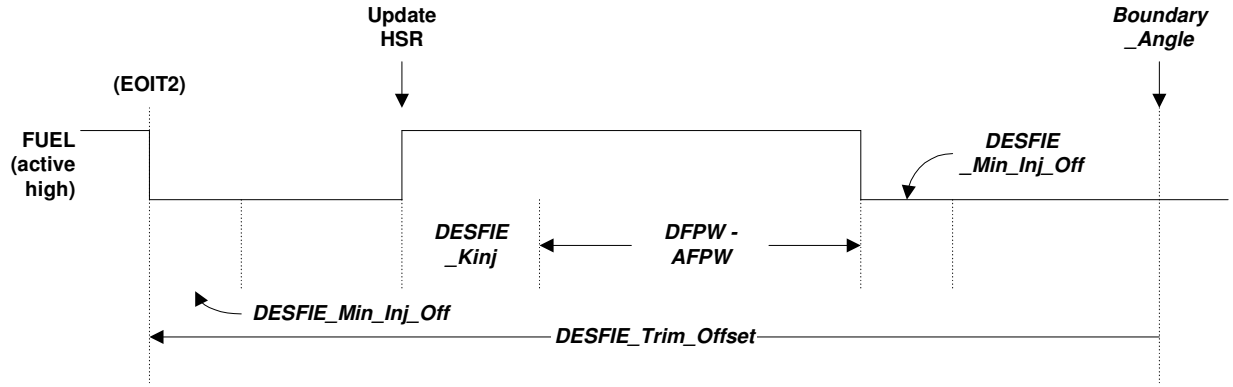


Figure 10 - Untargeted Trim Fuel Pulse

4.3.5 Simultaneous Fuel Delivery

To operate in simultaneous fuel delivery mode, the host CPU should perform the following steps:

1. Write values to ***Boundary_Angle*** (all fuel channels) and ***DESFIE_Normal_Offset*** such that the end-of-injection target (EOIT1) for each fuel channel is equal to (or less than) the current timer value.

2. Write the amount of desired fuel (***DFPW***) to each fuel channel.

3. Issue a **Boundary HSR** to each fuel channel.

Performing the above steps will cause untargeted fuel pulses to start at once on all fuel channels simultaneously.

There are a number of other possible methods for generating simultaneous fuel pulses, and each application must decide which method best suits its needs.

Handling the transition between simultaneous and sequential fuel delivery can be especially troublesome; the primary reason why the DESFIE function is designed to interrupt the host CPU at the end of every fuel pulse is to help with this transition. (Most applications will typically mask this interrupt during sequential fuel delivery).

Simultaneous fuel pulses are generated by the eTPU following the procedures described in **Sections** Error! Reference source not found. - **4.2.4**.

4.3.6 Individual Fuel Channel Disable

The host CPU can disable an individual fuel channel at any time by setting ***DFPW*** equal to 0x000000 (or any value less than or equal to ***AFPW***) and issuing either a **Boundary HSR** or an **Update HSR** (depending on circumstances). This will inhibit all fuel pulses for the current cylinder event, and cause any fuel pulse in progress to be truncated.

4.4 Update Host Service Request

The host CPU may issue an **Update Host Service Request** (%100) at any time between fuel boundaries. An **Update HSR** should be issued any time it is desired to change the value(s) of ***DFPW***, ***Boundary_Angle***, ***DESFIE_Normal_Offset***, ***DESFIE_Trim_Offset***, or ***DESFIE_Min_Trim*** (any or all of these may be updated at the same time). It is not necessary to issue an **Update HSR** to change ***DESFIE_Kinj*** or ***DESFIE_Min_Inj_Off*** (although they may be included as part of an **Update** if desired); simply writing new values to these variables is enough.

When an **Update HSR** is issued, the eTPU will use the new information to re-calculate and update any fuel pulse edge (leading or trailing) that may be pending, according to the procedures already described. Also, if fuel is being delivered when an **Update HSR** is issued (ie, the fuel pulse is on and ***DESFIE_Kinj*** is not in progress), the eTPU will update ***AFPW*** immediately. (The current timer value will be stored in ***Start_End_Time*** in this case.)

If an **Update HSR** is issued when ***Kinj_Active*** = 1, the new information will be ignored until ***Kinj_Active*** = 0. The same applies to ***MIO_Active***. Therefore, neither ***DESFIE_Kinj*** nor ***DESFIE_Min_Inj_Off*** can be changed while they are in progress.

4.5 Shutdown Host Service Request

The DESFIE function can be disabled by issuing a **Shutdown Host Service Request** (%111), waiting for the HSR bits to clear, and setting the priority level to disabled (%00).

> **NOTE: If *Active_High* = 0 and no fuel pulse was in progress at the time of the HSR, a low glitch lasting approximately 2 microcycles will occur on the output pin (see the Application Implementation document to convert microcycles into real time based on crystal frequency).**

4.6 Link From Another eTPU Channel

The DESFIE function will respond to a link from another eTPU channel by performing an **Update HSR** (see **Section 4.4**).

4.7 Switching from DESFIE to Another eTPU Function

To switch from the DESFIE function to another eTPU function, the host CPU should shut down the DESFIE function (see **Section 4.5**) and then follow the directions for initializing the new function.

4.8 Accuracy

4.8.1 Requested vs. Delivered Fuel

The amount of fuel requested for a given fuel channel during a given cylinder event (***DFPW***) will be delivered, except in the following cases:

1. Fuel delivery is disabled by the method described in **Section 4.3.6**.
2. Trim pulses are disabled by setting ***DESFIE_Min_Trim*** = 0xFFFFFF and/or setting ***Multiple_Trim***.
3. A fuel pulse is truncated at a boundary.

4.8.2 Accuracy of Delivered Fuel

The calculated amount of delivered fuel (***AFPW***) will be accurate unless ***DESFIE_Min_Inj_Off*** is violated due to one of the boundary rules (see **Section 4.1.1**).

4.9 Latency

The worst-case execution times for the various threads of the DESFIE function are shown in **Table 2** (see the Application Implementation document to convert microcycles into real time based on crystal frequency). Note that these values do not take into account latencies due to other functions in the application, priorities, etc.

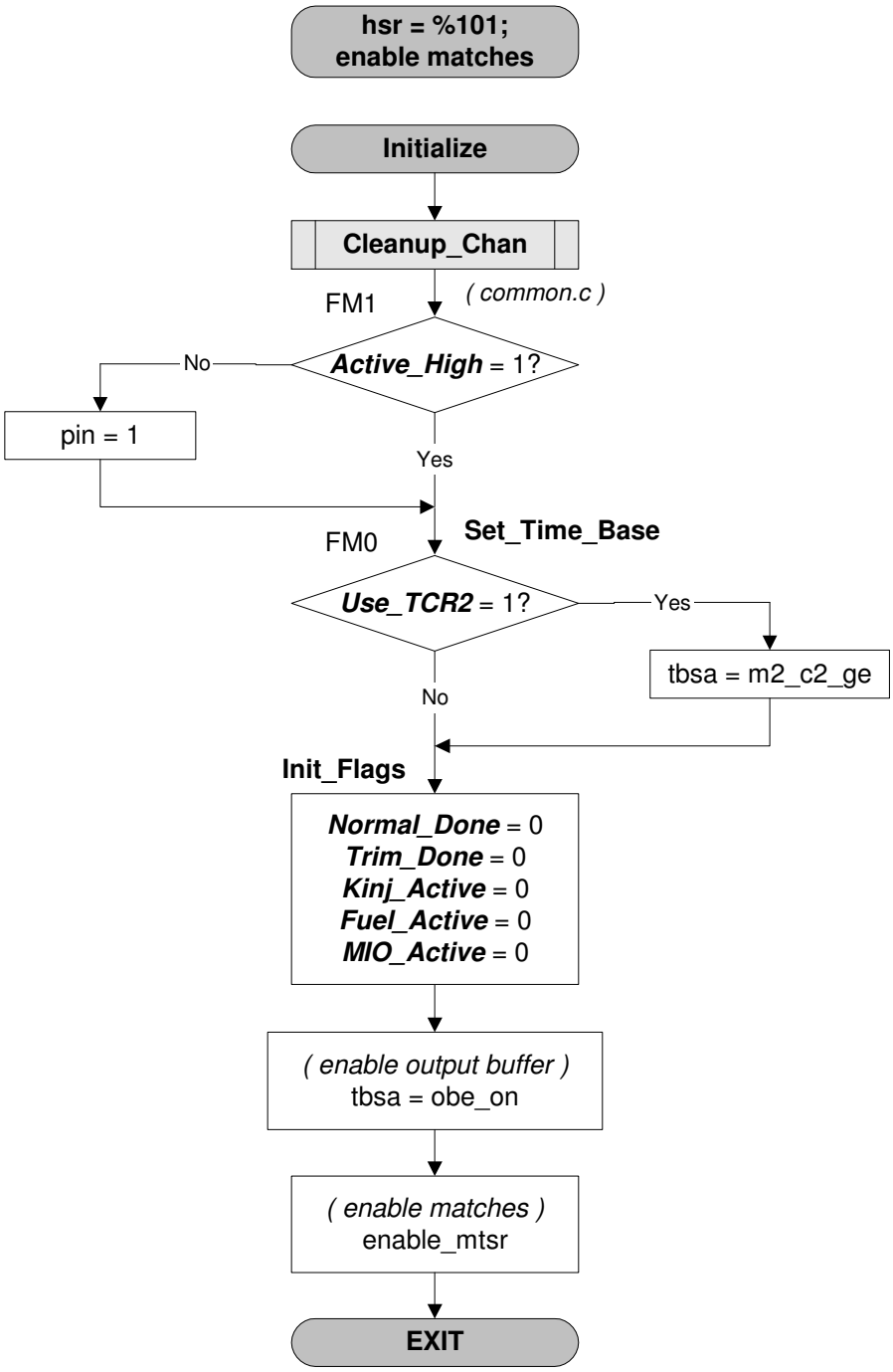
DESFIE Thread	Worst-Case µcycles *	RAM Accesses
Initialize HSR	**15**	**1**
Shutdown HSR	**10**	**0**
Boundary HSR	**107**	**15**
Update HSR	**80**	**11**
Link	**82**	**11**
Output Event:		
Leading edge of fuel pulse	**27**	**4**
End of Kinj	**25**	**2**
Trailing edge of fuel pulse	**23**	**6**
End of Min_Inj_Off	**77**	**12**

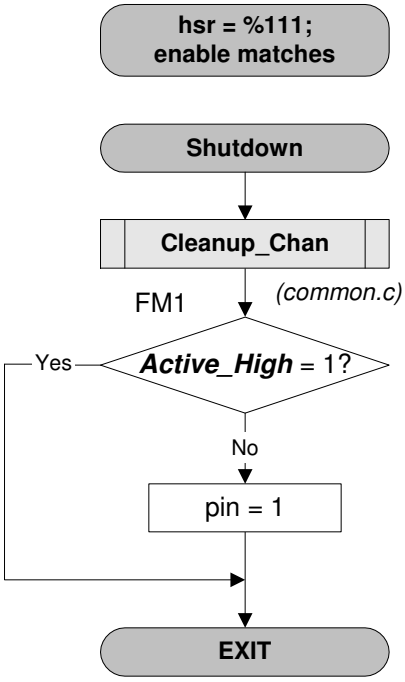
Table 2 - Latencies

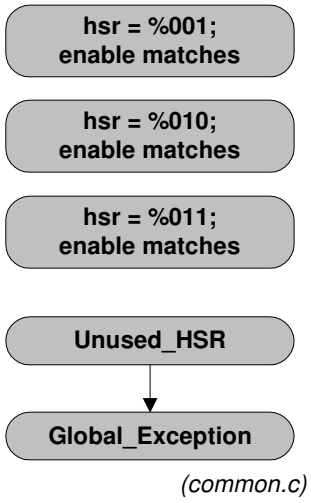
Total number of instructions: 225

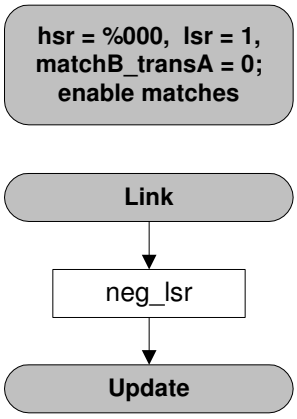
* These numbers do not consider potential collisions while accessing RAM.

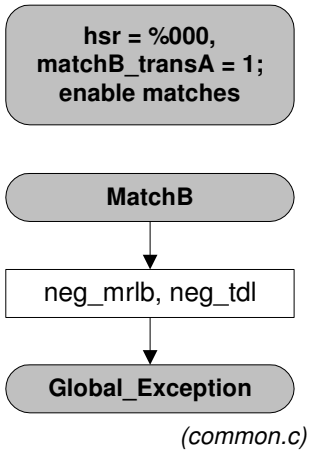
5. Flowcharts

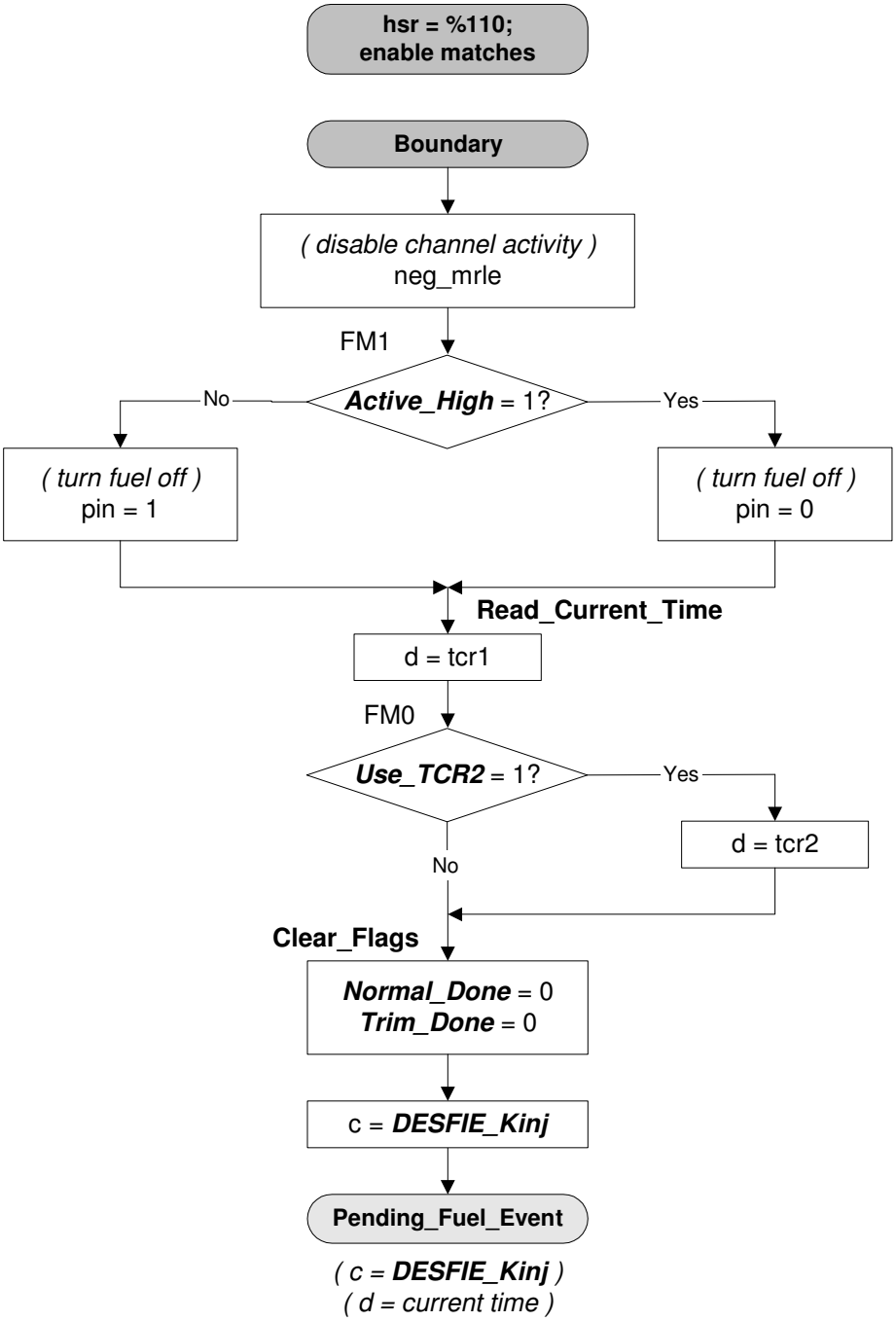


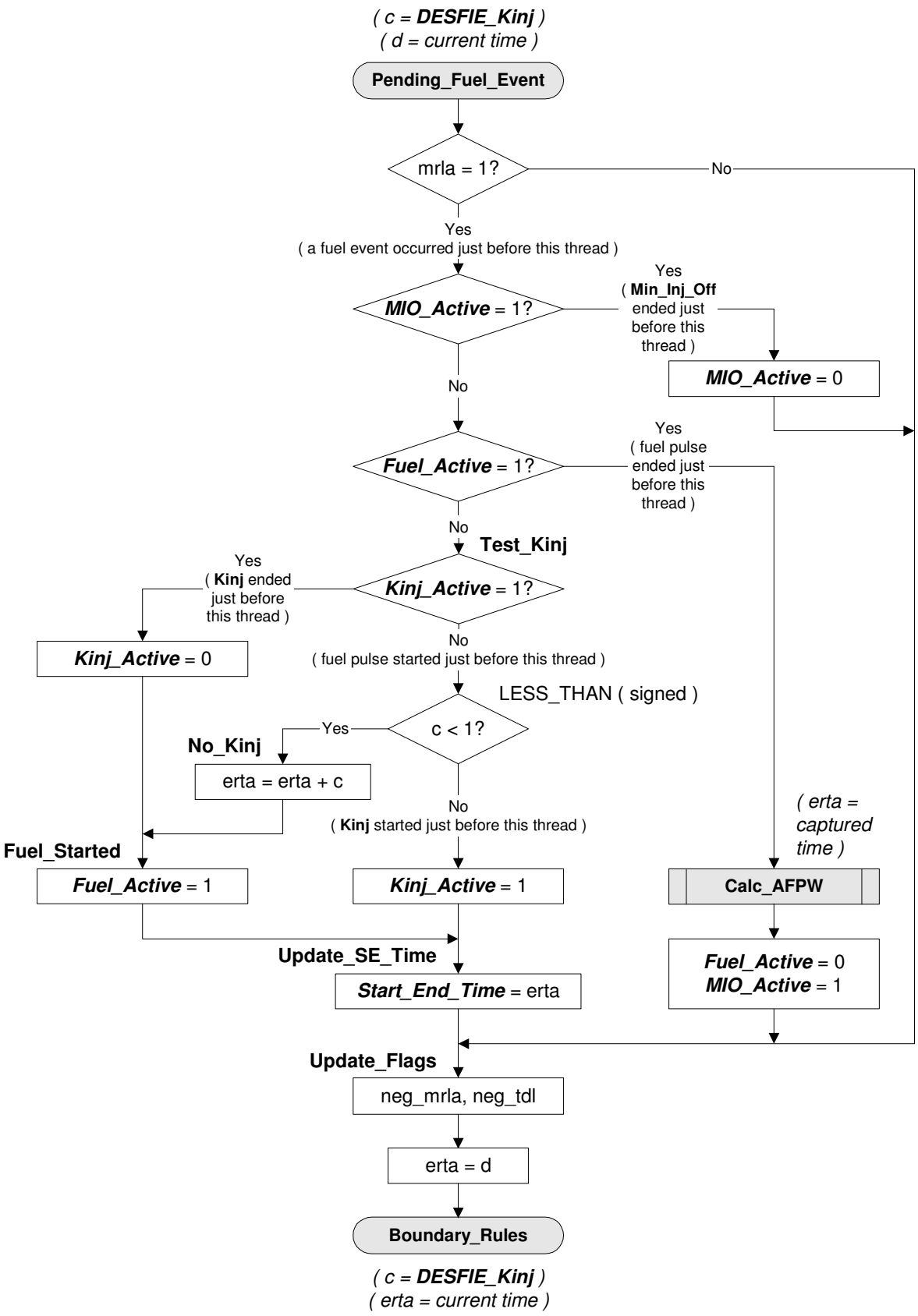

hsr = %111;
enable matches
Shutdown
Cleanup_Chan
(common.c)
FM1
Yes
Active_High = 1?
No
pin = 1
EXIT

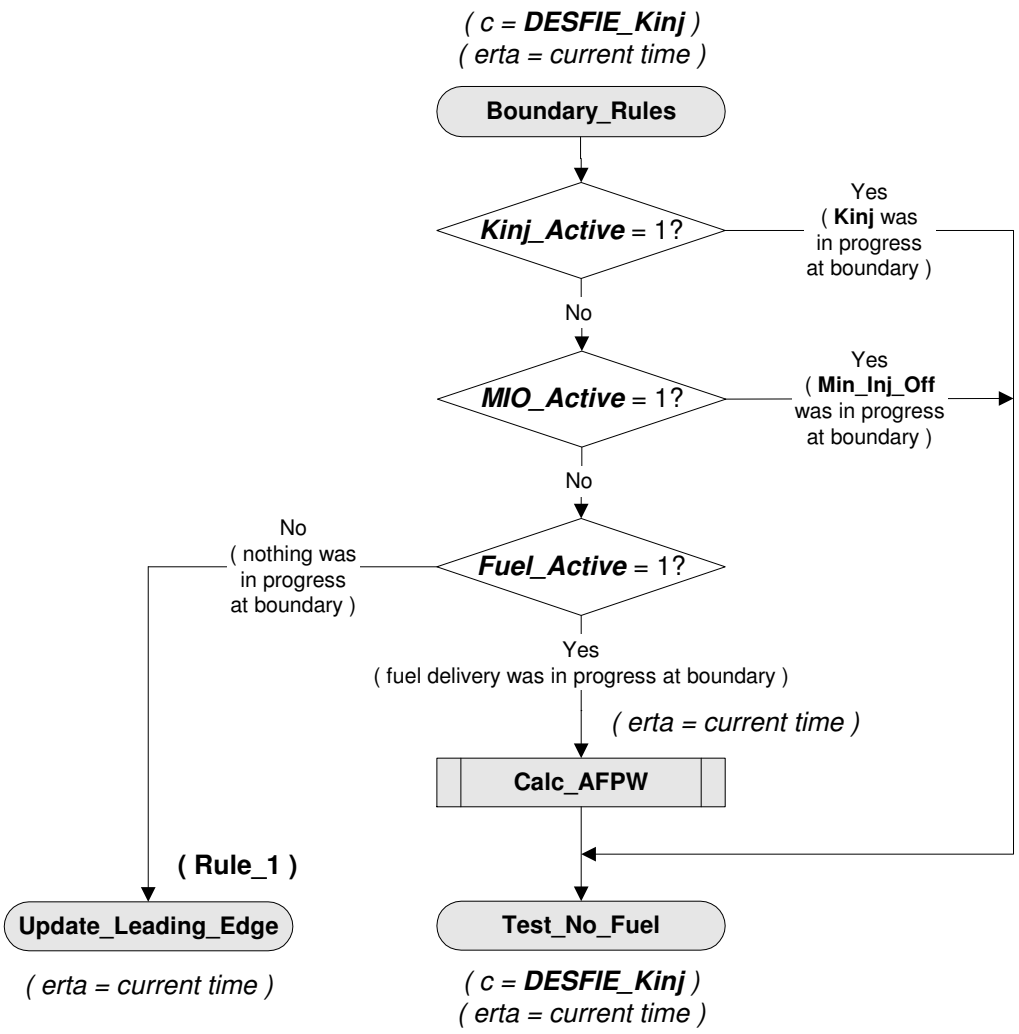

hsr = %001;
enable matches
hsr = %010;
enable matches
hsr = %011;
enable matches
Unused_HSR
Global_Exception
(common.c)

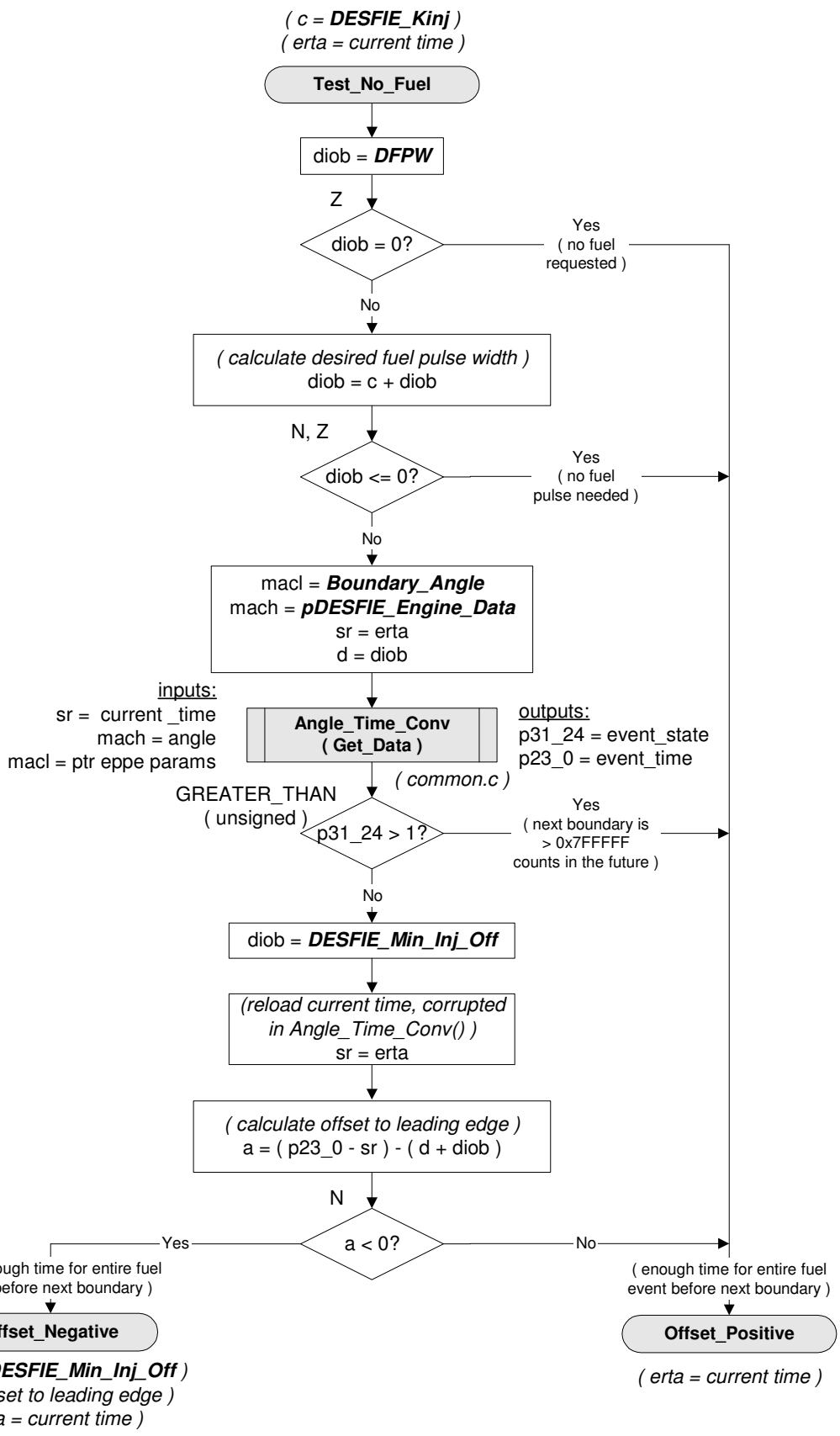

hsr = %000, lsr = 1,
matchB_transA = 0;
enable matches
Link
neg_lsr
Update

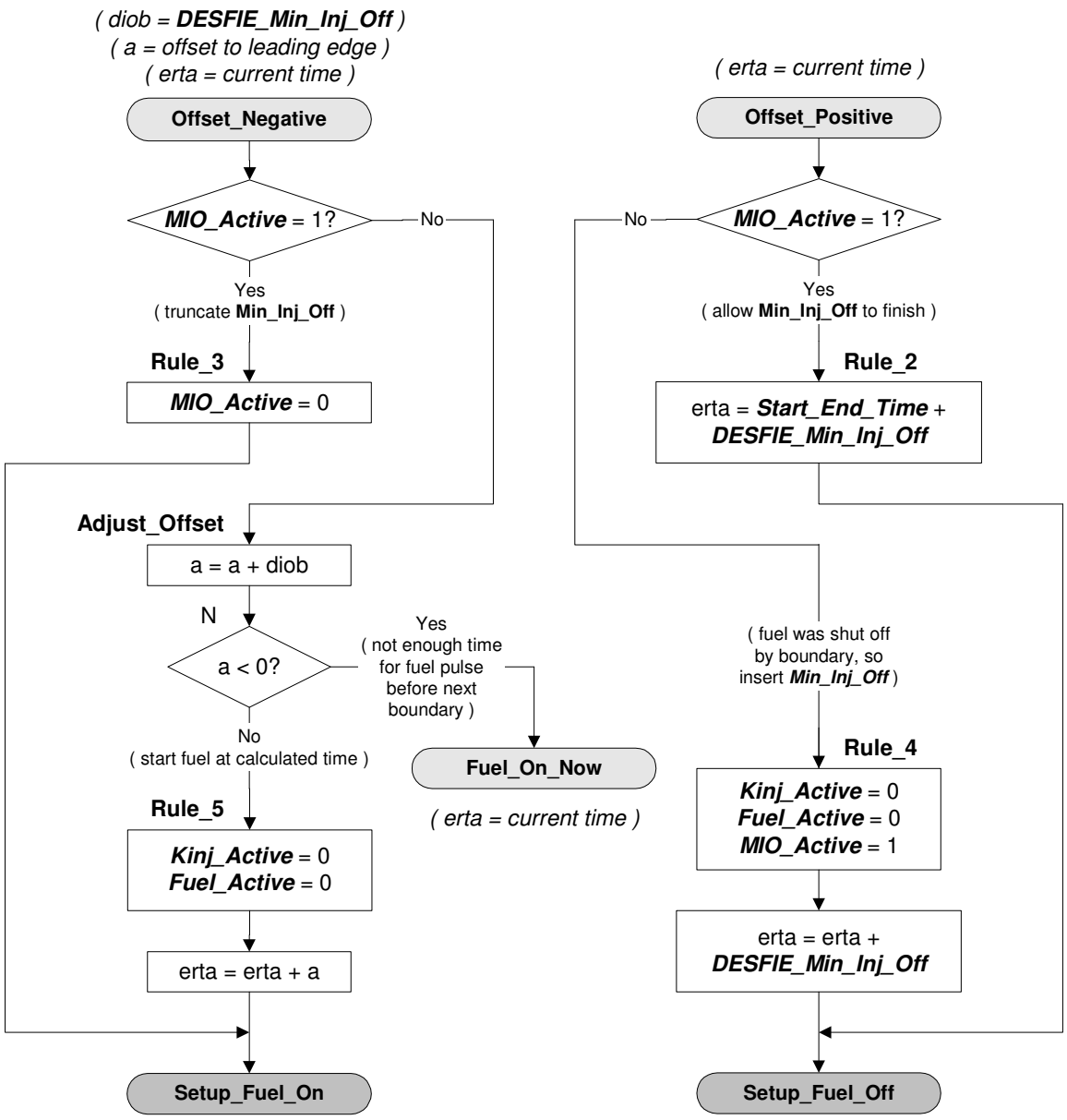

hsr = %000,
matchB_transA = 1;
enable matches
MatchB
neg_mrlb, neg_tdl
Global_Exception
(common.c)

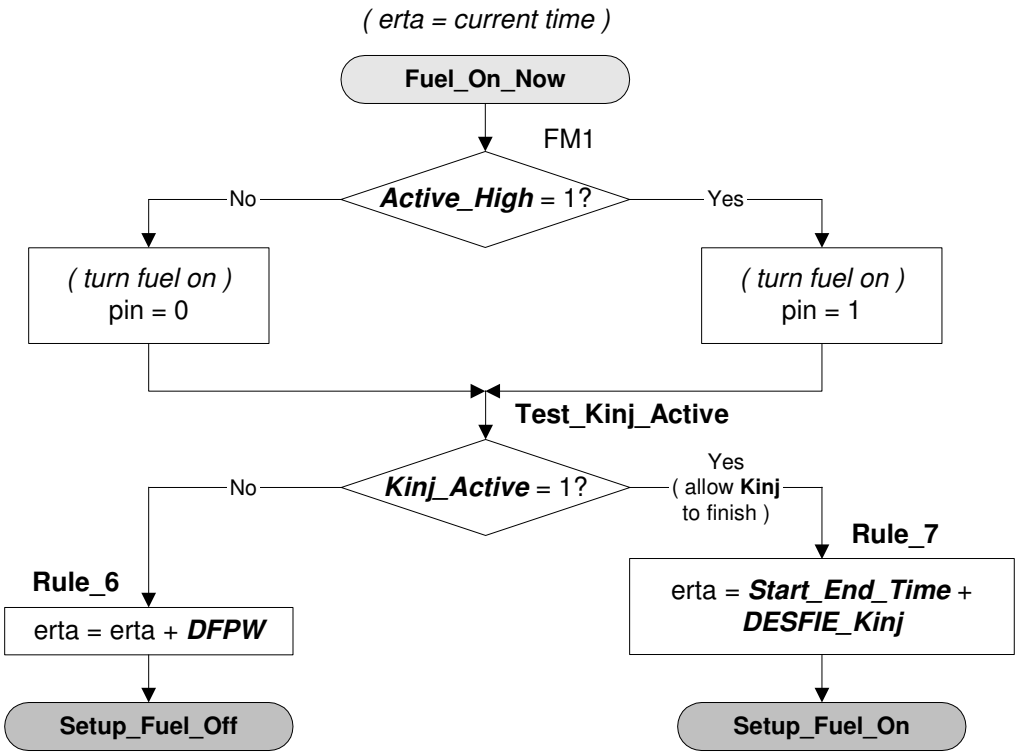

hsr = %110;
enable matches
Boundary
(disable channel activity)
neg_mrle
FM1
Active_High = 1?
No
Yes
(turn fuel off)
pin = 1
(turn fuel off)
pin = 0
Read_Current_Time
d = tcr1
FM0
Use_TCR2 = 1?
Yes
d = tcr2
No
Clear_Flags
Normal_Done = 0
Trim_Done = 0
c = DESFIE_Kinj
Pending_Fuel_Event
(c = DESFIE_Kinj)
(d = current time)

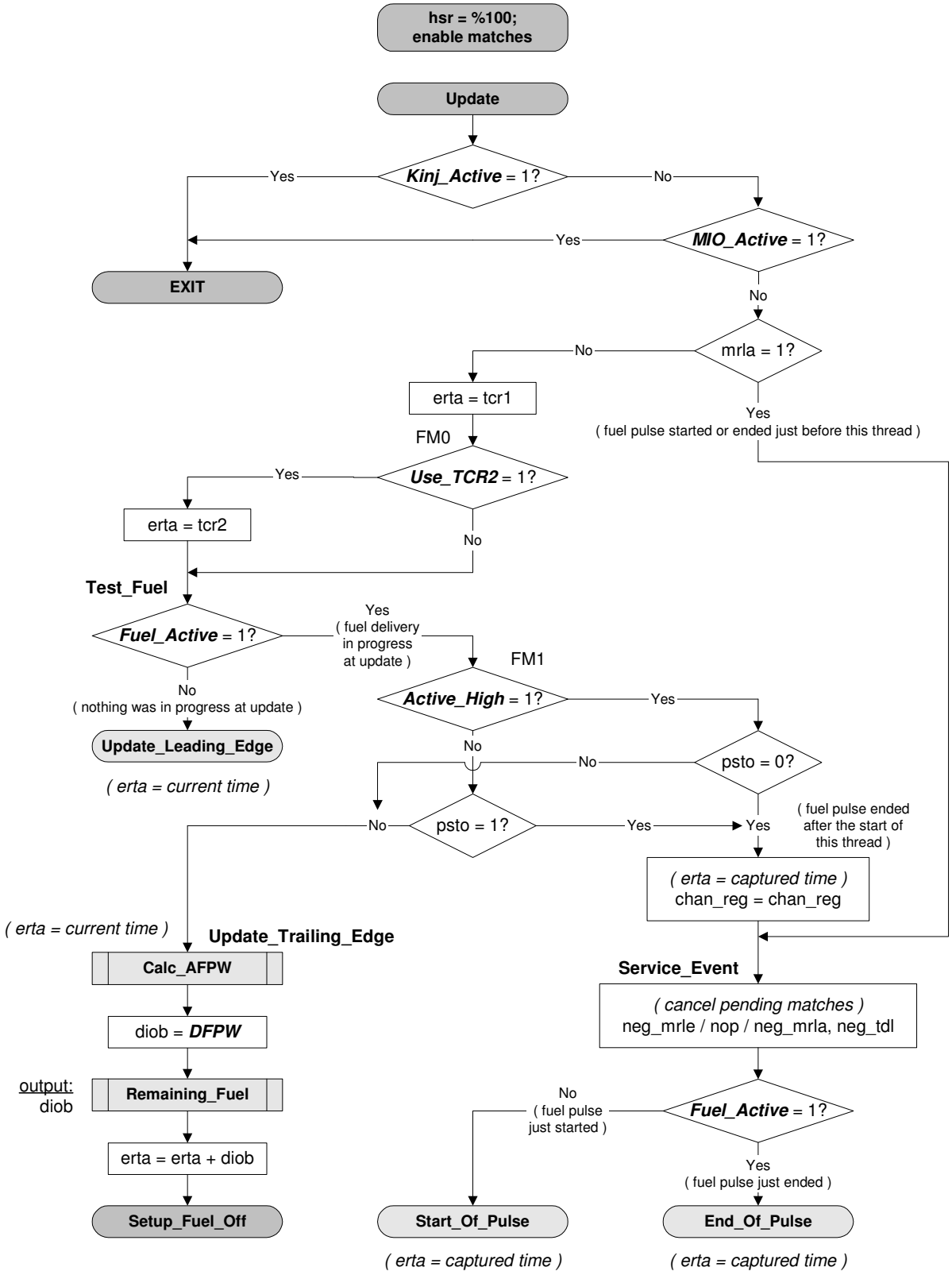

(c = DESFIE_Kinj)
(d = current time)
Pending_Fuel_Event
mrla = 1?
No
Yes
(a fuel event occurred just before this thread)
MIO_Active = 1?
Yes
(Min_Inj_Off ended just before this thread)
MIO_Active = 0
No
Fuel_Active = 1?
Yes
(fuel pulse ended just before this thread)
No
Test_Kinj
Kinj_Active = 1?
Yes
(Kinj ended just before this thread)
Kinj_Active = 0
No
(fuel pulse started just before this thread)
LESS_THAN (signed)
c < 1?
Yes
No_Kinj
erta = erta + c
No
(Kinj started just before this thread)
(erta = captured time)
Fuel_Started
Fuel_Active = 1
Kinj_Active = 1
Calc_AFPW
Update_SE_Time
Start_End_Time = erta
Fuel_Active = 0
MIO_Active = 1
Update_Flags
neg_mrla, neg_tdl
erta = d
Boundary_Rules
(c = DESFIE_Kinj)
(erta = current time)

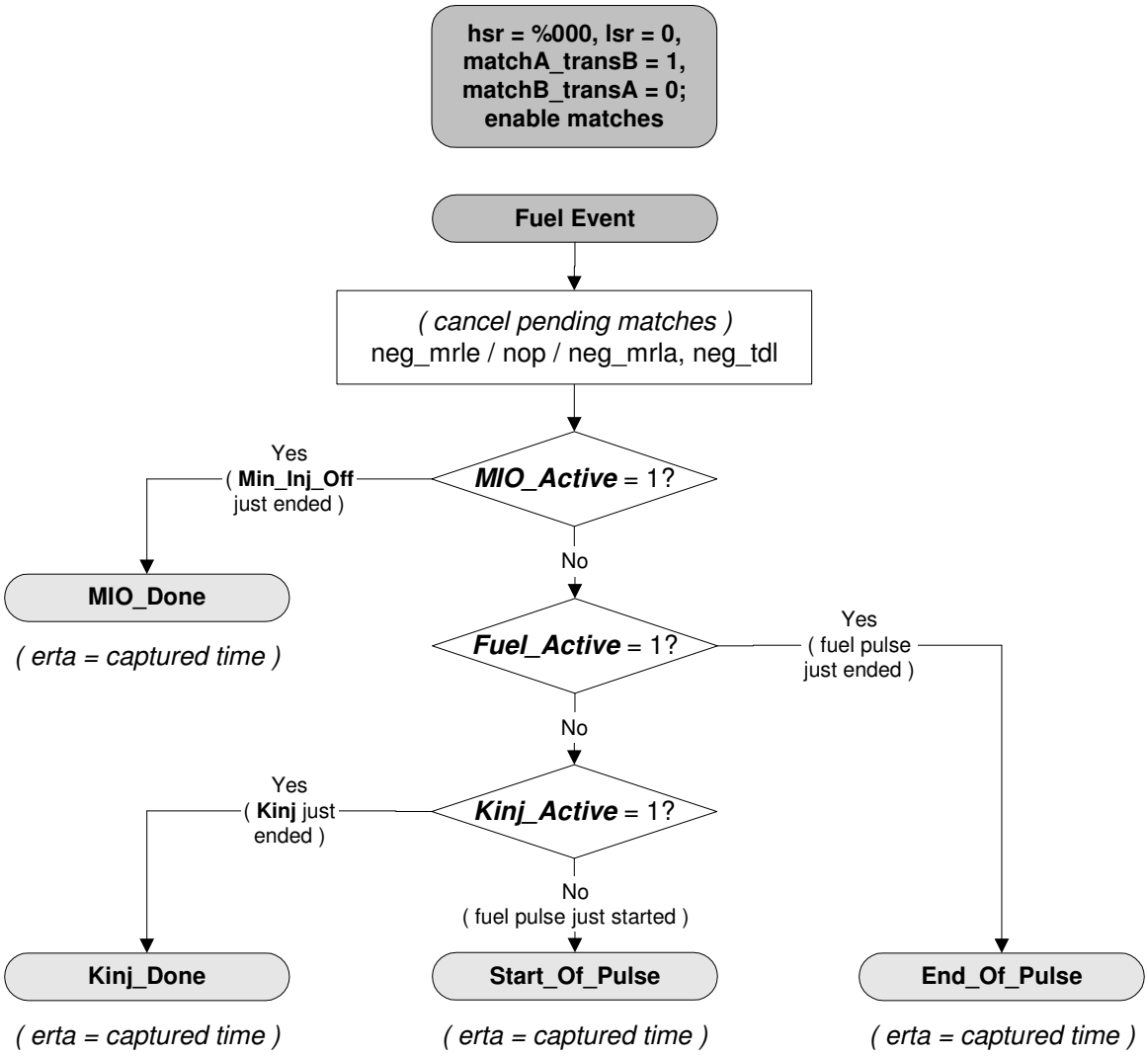

(c = DESFIE_Kinj)
(erta = current time)
Boundary_Rules
Kinj_Active = 1?
Yes
(Kinj was
in progress
at boundary)
No
MIO_Active = 1?
Yes
(Min_Inj_Off
was in progress
at boundary)
No
Fuel_Active = 1?
No
(nothing was
in progress
at boundary)
Yes
(fuel delivery was in progress at boundary)
(erta = current time)
Calc_AFPW
(Rule_1)
Update_Leading_Edge
(erta = current time)
Test_No_Fuel
(c = DESFIE_Kinj)
(erta = current time)

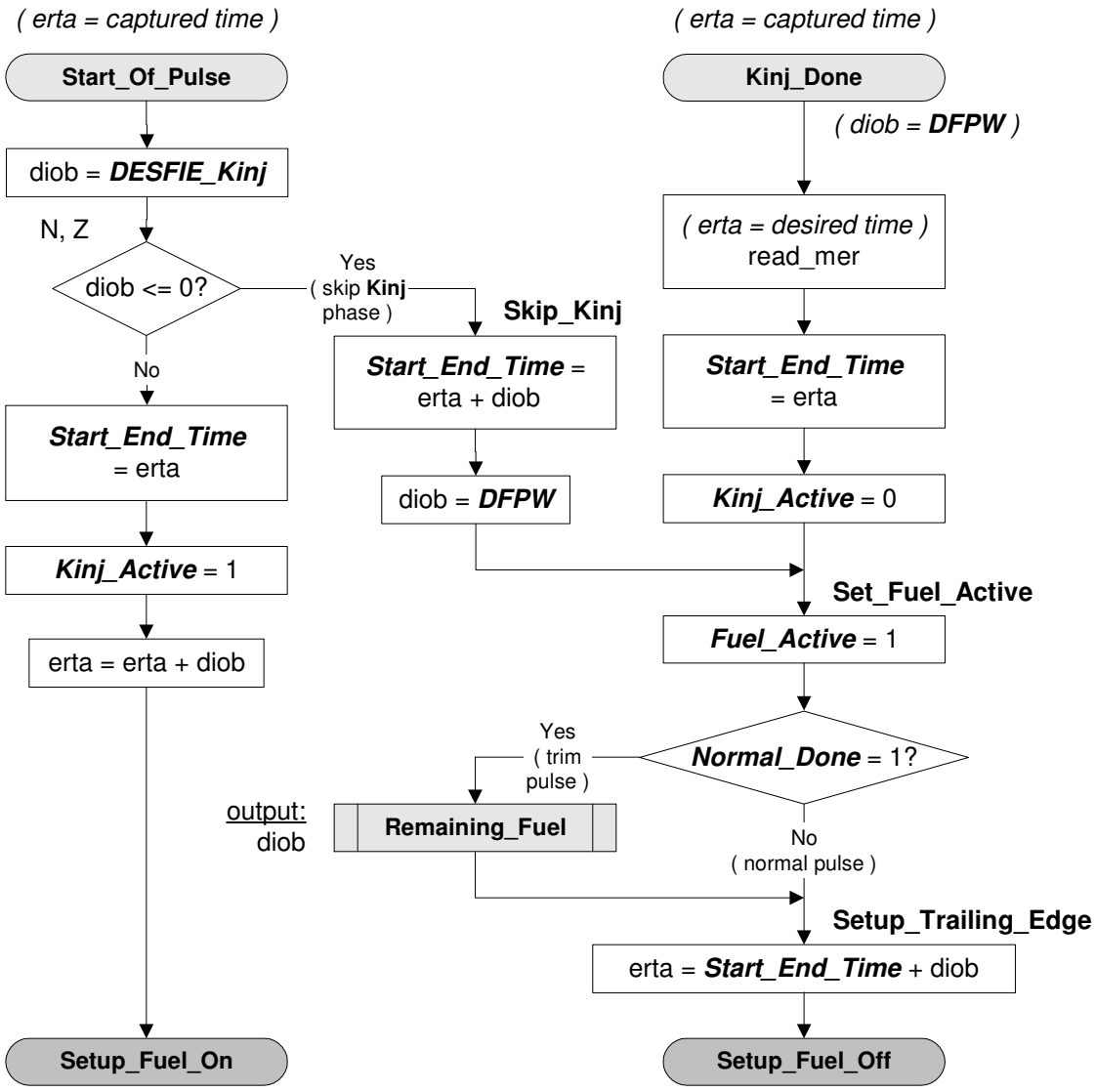

(c = DESFIE_Kinj)
(erta = current time)
Test_No_Fuel
diob = DFPW
Z
diob = 0?
Yes
(no fuel
requested)
No
(calculate desired fuel pulse width)
diob = c + diob
N, Z
diob <= 0?
Yes
(no fuel
pulse needed)
No
macl = Boundary_Angle
mach = pDESFIE_Engine_Data
sr = erta
d = diob
inputs:
sr = current _time
mach = angle
macl = ptr eppe params
Angle_Time_Conv
(Get_Data)
outputs:
p31_24 = event_state
p23_0 = event_time
(common.c)
GREATER_THAN
(unsigned)
p31_24 > 1?
Yes
(next boundary is
> 0x7FFFFF
counts in the future)
No
diob = DESFIE_Min_Inj_Off
(reload current time, corrupted
in Angle_Time_Conv())
sr = erta
(calculate offset to leading edge)
a = (p23_0 - sr) - (d + diob)
N
a < 0?
Yes
(not enough time for entire fuel
event before next boundary)
No
(enough time for entire fuel
event before next boundary)
Offset_Negative
(diob = DESFIE_Min_Inj_Off)
(a = offset to leading edge)
(erta = current time)
Offset_Positive
(erta = current time)

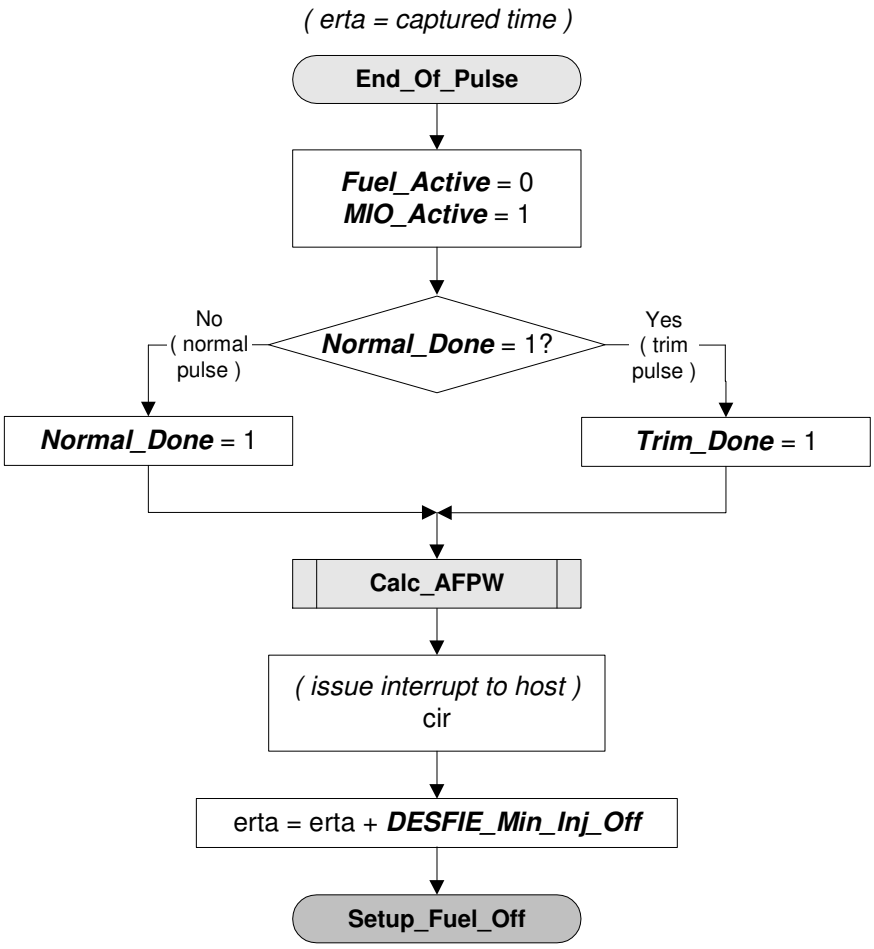

(diob = DESFIE_Min_Inj_Off)
(a = offset to leading edge)
(erta = current time)
Offset_Negative
MIO_Active = 1?
No
Yes
(truncate Min_Inj_Off)
Rule_3
MIO_Active = 0
Adjust_Offset
a = a + diob
N
a < 0?
Yes
(not enough time for fuel pulse before next boundary)
No
(start fuel at calculated time)
Fuel_On_Now
(erta = current time)
Rule_5
Kinj_Active = 0
Fuel_Active = 0
erta = erta + a
Setup_Fuel_On
(erta = current time)
Offset_Positive
No
MIO_Active = 1?
Yes
(allow Min_Inj_Off to finish)
Rule_2
erta = Start_End_Time + DESFIE_Min_Inj_Off
(fuel was shut off by boundary, so insert Min_Inj_Off)
Rule_4
Kinj_Active = 0
Fuel_Active = 0
MIO_Active = 1
erta = erta + DESFIE_Min_Inj_Off
Setup_Fuel_Off

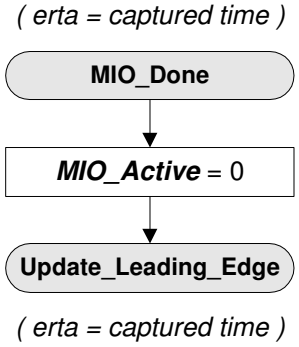

(erta = current time)
Fuel_On_Now
FM1
Active_High = 1?
No
Yes
(turn fuel on)
pin = 0
(turn fuel on)
pin = 1
Test_Kinj_Active
Kinj_Active = 1?
No
Yes
(allow Kinj
to finish)
Rule_6
erta = erta + DFPW
Rule_7
erta = Start_End_Time +
DESFIE_Kinj
Setup_Fuel_Off
Setup_Fuel_On

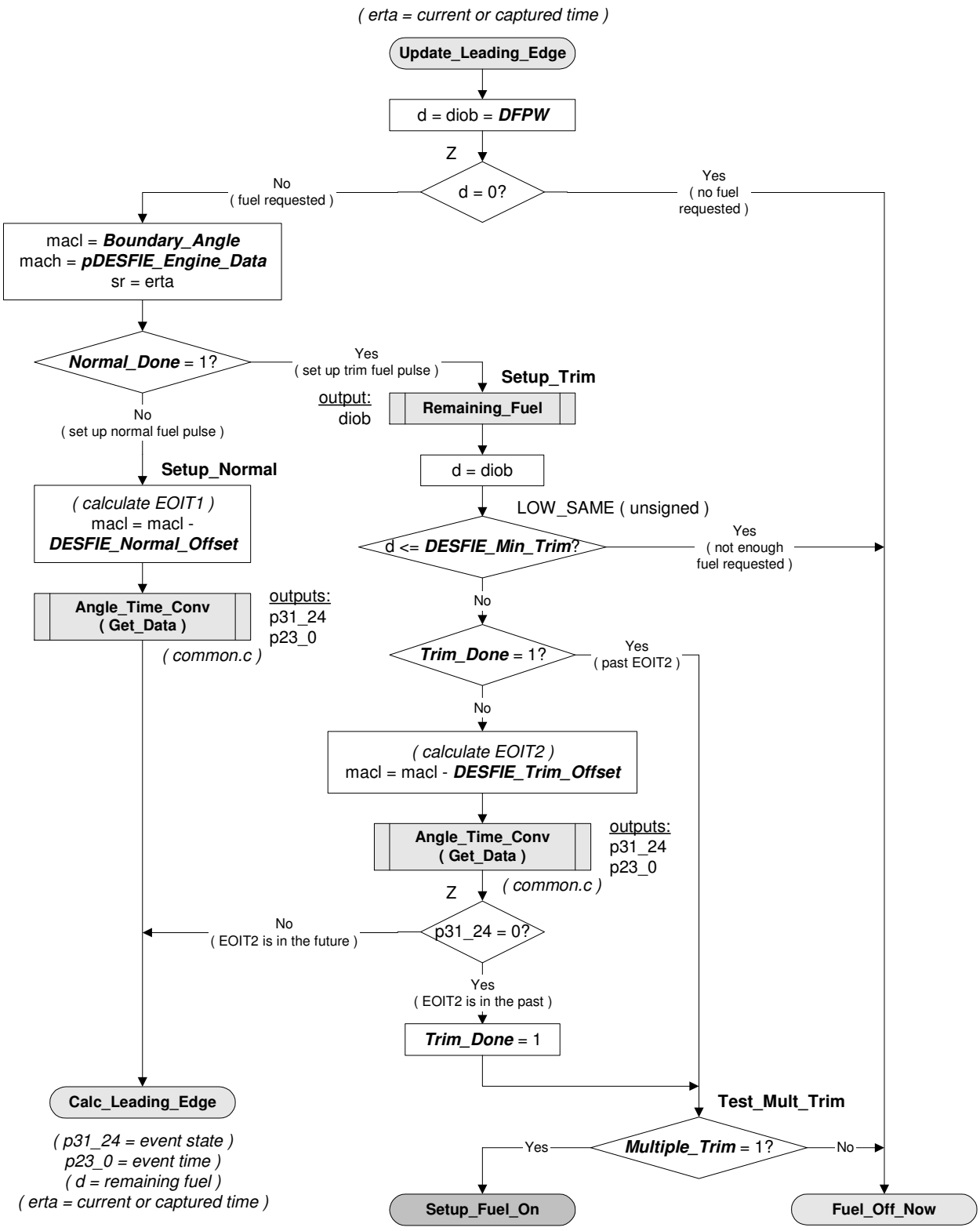

hsr = %100;
enable matches
Update
Kinj_Active = 1?
Yes
No
MIO_Active = 1?
Yes
EXIT
No
mrla = 1?
No
Yes
(fuel pulse started or ended just before this thread)
erta = tcr1
FM0
Use_TCR2 = 1?
Yes
erta = tcr2
No
Test_Fuel
Fuel_Active = 1?
Yes
(fuel delivery in progress at update)
No
(nothing was in progress at update)
Update_Leading_Edge
(erta = current time)
FM1
Active_High = 1?
Yes
No
psto = 0?
No
psto = 1?
No
Yes
Yes
(fuel pulse ended after the start of this thread)
(erta = captured time)
chan_reg = chan_reg
(erta = current time)
Update_Trailing_Edge
Calc_AFPW
diob = DFPW
output:
diob
Remaining_Fuel
erta = erta + diob
Setup_Fuel_Off
Service_Event
(cancel pending matches)
neg_mrle / nop / neg_mrla, neg_tdl
Fuel_Active = 1?
No
(fuel pulse just started)
Yes
(fuel pulse just ended)
Start_Of_Pulse
(erta = captured time)
End_Of_Pulse
(erta = captured time)

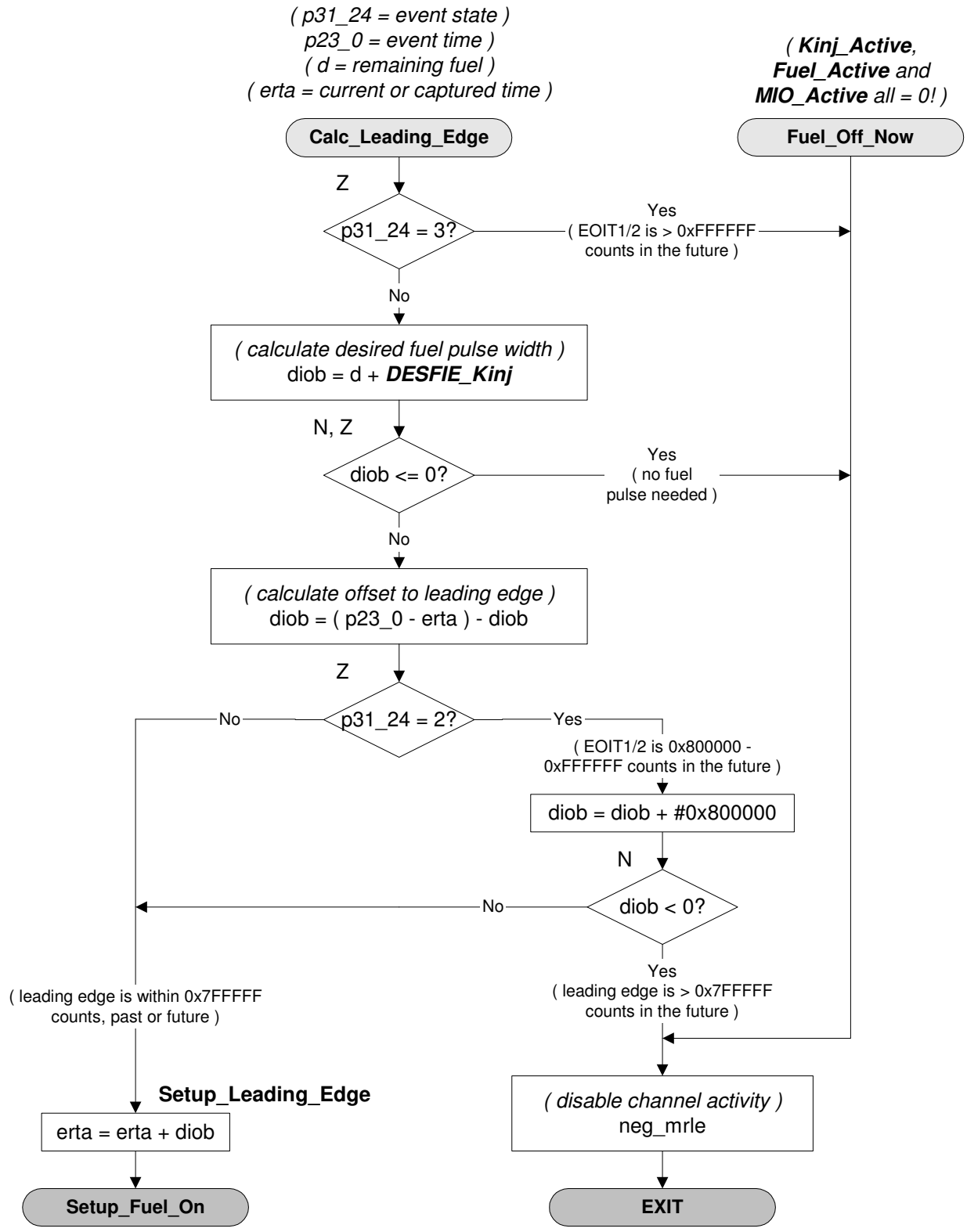

hsr = %000, lsr = 0,
matchA_transB = 1,
matchB_transA = 0;
enable matches
Fuel Event
(cancel pending matches)
neg_mrle / nop / neg_mrla, neg_tdl
MIO_Active = 1?
Yes
(Min_Inj_Off just ended)
No
MIO_Done
(erta = captured time)
Fuel_Active = 1?
Yes
(fuel pulse just ended)
No
Kinj_Active = 1?
Yes
(Kinj just ended)
No
(fuel pulse just started)
Kinj_Done
(erta = captured time)
Start_Of_Pulse
(erta = captured time)
End_Of_Pulse
(erta = captured time)


(erta = captured time)
Start_Of_Pulse
diob = DESFIE_Kinj
N, Z
diob <= 0?
Yes
(skip Kinj phase)
Skip_Kinj
Start_End_Time = erta + diob
diob = DFPW
No
Start_End_Time = erta
Kinj_Active = 1
erta = erta + diob
Setup_Fuel_On
(erta = captured time)
Kinj_Done
(diob = DFPW)
(erta = desired time)
read_mer
Start_End_Time = erta
Kinj_Active = 0
Set_Fuel_Active
Fuel_Active = 1
Yes
(trim pulse)
Normal_Done = 1?
output:
diob
Remaining_Fuel
No
(normal pulse)
Setup_Trailing_Edge
erta = Start_End_Time + diob
Setup_Fuel_Off

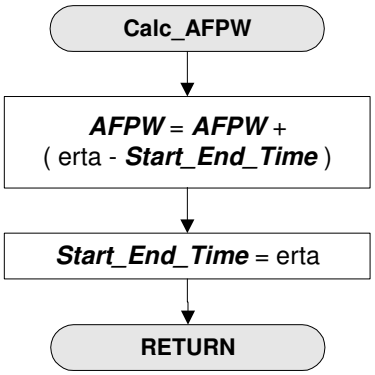

(erta = captured time)
End_Of_Pulse
Fuel_Active = 0
MIO_Active = 1
No
(normal
pulse)
Normal_Done = 1?
Yes
(trim
pulse)
Normal_Done = 1
Trim_Done = 1
Calc_AFPW
(issue interrupt to host)
cir
erta = erta + DESFIE_Min_Inj_Off
Setup_Fuel_Off


(erta = captured time)
MIO_Done
MIO_Active = 0
Update_Leading_Edge
(erta = captured time)

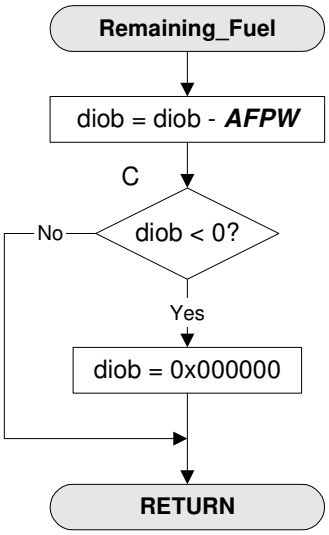

(erta = current or captured time)
Update_Leading_Edge
d = diob = DFPW
Z
d = 0?
No
(fuel requested)
Yes
(no fuel
requested)
macl = Boundary_Angle
mach = pDESFIE_Engine_Data
sr = erta
Normal_Done = 1?
Yes
(set up trim fuel pulse)
Setup_Trim
output:
diob
Remaining_Fuel
No
(set up normal fuel pulse)
Setup_Normal
(calculate EOIT1)
macl = macl -
DESFIE_Normal_Offset
d = diob
LOW_SAME (unsigned)
d <= DESFIE_Min_Trim?
Yes
(not enough
fuel requested)
Angle_Time_Conv
(Get_Data)
outputs:
p31_24
p23_0
(common.c)
No
Trim_Done = 1?
Yes
(past EOIT2)
No
(calculate EOIT2)
macl = macl - DESFIE_Trim_Offset
Angle_Time_Conv
(Get_Data)
outputs:
p31_24
p23_0
(common.c)
Z
p31_24 = 0?
No
(EOIT2 is in the future)
Yes
(EOIT2 is in the past)
Trim_Done = 1
Calc_Leading_Edge
Test_Mult_Trim
Multiple_Trim = 1?
Yes
No
(p31_24 = event state)
p23_0 = event time)
(d = remaining fuel)
(erta = current or captured time)
Setup_Fuel_On
Fuel_Off_Now


(p31_24 = event state)
p23_0 = event time)
(d = remaining fuel)
(erta = current or captured time)
(Kinj_Active, Fuel_Active and MIO_Active all = 0!)
Calc_Leading_Edge
Fuel_Off_Now
Z
p31_24 = 3?
Yes
(EOIT1/2 is > 0xFFFFFF counts in the future)
No
(calculate desired fuel pulse width)
diob = d + DESFIE_Kinj
N, Z
diob <= 0?
Yes
(no fuel pulse needed)
No
(calculate offset to leading edge)
diob = (p23_0 - erta) - diob
Z
p31_24 = 2?
No
Yes
(EOIT1/2 is 0x800000 - 0xFFFFFF counts in the future)
diob = diob + #0x800000
N
diob < 0?
No
Yes
(leading edge is within 0x7FFFFF counts, past or future)
(leading edge is > 0x7FFFFF counts in the future)
Setup_Leading_Edge
erta = erta + diob
(disable channel activity)
neg_mrle
Setup_Fuel_On
EXIT

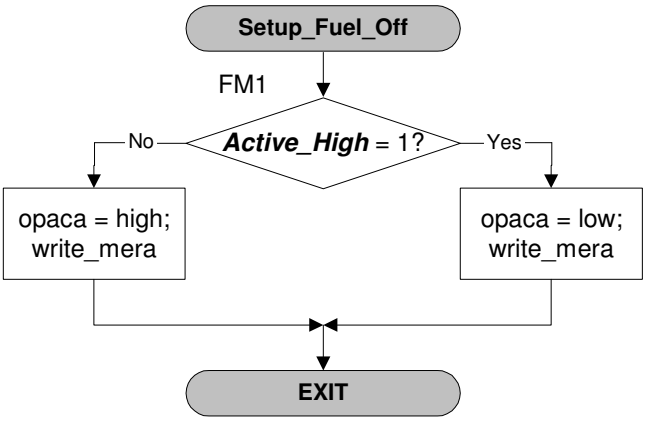
(erta = current or captured time)

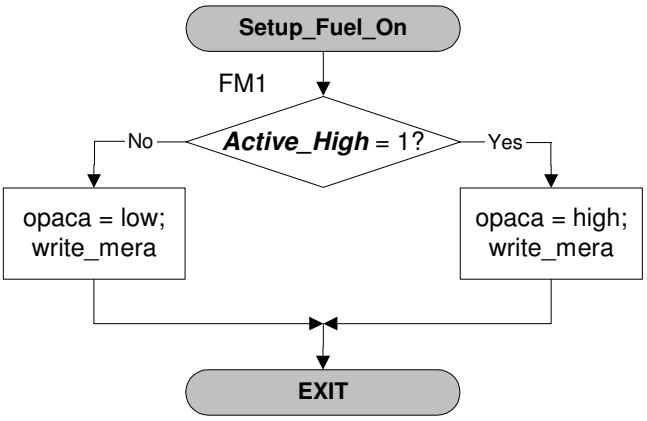


*(diob = **DFPW**)*



(diob = remaining fuel)





6. Revision History

6.1 Document Revision Numbering

Each microcode document is assigned a revision number. The numbers are assigned according to the following scheme (used for all documents after May, 2003):

Rev x.y

x identifies the microcode, where

1 represents the original release of microcode

2 represents the first release of changed microcode

3 represents the second change to microcode etc.

y identifies the document, where

0 represents the original release of documentation for this microcode

1 represents the first document change for the same microcode

2 represents the second change to the document for the same microcode

6.2 Revision Log

Revision	Date	Record	Author
1.0	09-02-05	Initial release, based on DESFIE-4.1 (eTPU). Changed ***Boundary_Angle***, ***Normal_Offset***, and ***Trim_Offset*** to angular values. Used separate flags for **Kinj** and **Min_Inj_Off**, and added ***Fuel_Active*** flag. Allowed for ***DESFIE_Kinj*** to be a negative value. Set ***Trim_Done*** flag when EOIT2 is passed, whether or not a targeted trim pulse was delivered.	Warren Donley
2.0 SCR #8637	03-17-09	Fixed misplaced fuel pulse issue caused by SR being corrupted in ATC_Get_Data().	Mary Hedges