PS700Driver C

Battery Monitor Driver C Code

1.0 PRODUCT OVERVIEW

The PS700Driver C is the code for the PS700 battery monitor driver written in the C programming language. It is patterned after the PS700Driver, written in PIC16F compatible assembly language. PS700Driver C is designed to be portable with limited system dependencies. Final implementation and integration requires a custom API for:

- Execution of the fuel gauge functions
- Data exchange
- Communication with the PS700

The PS700Driver C is a single C language source file developed in the Microsoft® Visual C++® environment.

Efficient communication is provided through an industry standard SMBus/I^{2}C™ compatible 2-wire communications interface. This interface allows the system to determine accurate battery status for effective system power management and for communication to the end user. A battery management solution, utilizing the PS700, delivers both space and total system component cost savings for a wide variety of battery operated applications.

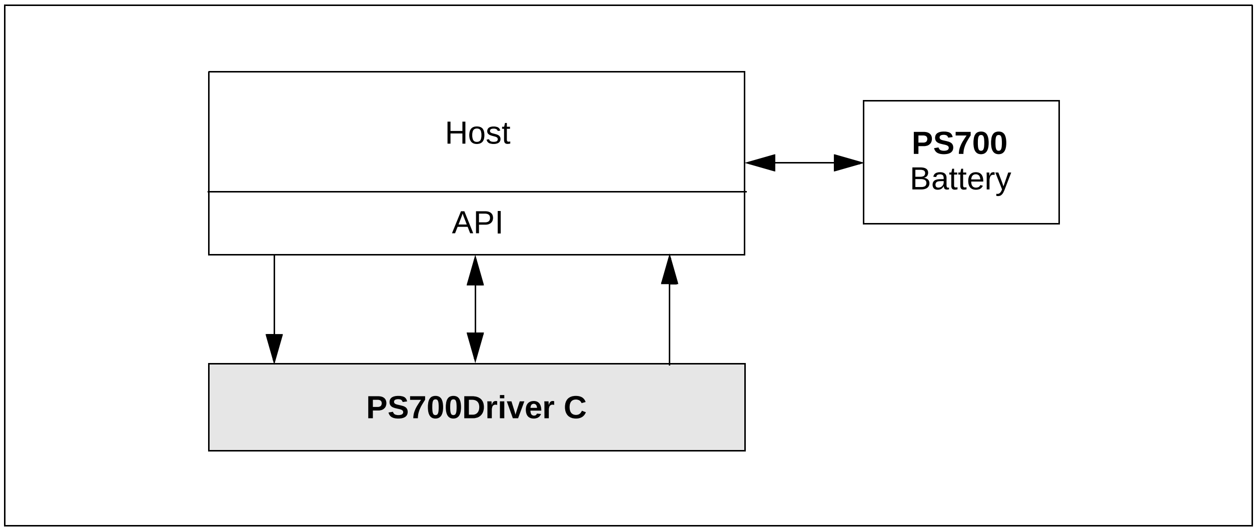
2.0 PS700DRIVER C ARCHITECTURE

The PS700Driver C is a block of C programming language code providing the core fuel gauge functionality for the PS700. The PS700Driver C is a module that resides in the application and is callable by the application's primary firmware. The PS700Driver C interacts with the host using:

1. Calls by the host to PS700Driver C functions.
2. Data exchange through global variables (common memory).
3. Calls to the host by PS700Driver C for SMBus communication.

Figure 2-1 is a block diagram of this architecture.

FIGURE 2-1: PS700DRIVER C ARCHITECTURE EXAMPLE



2.1 Code Organization

2.1.1 FILES

- `p7fgc.cpp` – fuel gauge, c code
- `p7fgc.h` – fuel gauge, header file
- `p7fgc_typedef.h` – data type definitions
- `p7fgc_dev.h` – memory map definitions for PS700 hardware registers

2.1.2 DATA TYPES

A header file is used for system independent type definitions. The following file is used for the Microsoft Visual C++ on a PC.

EXAMPLE 2-1: FILE: `p7fgc_typedef.h`

```
//--------------------------------------
//--- type definitions
//--------------------------------------
#ifndef _P7FGC_TYPEDEF_H
#define _P7FGC_TYPEDEF_H
typedef char           TD_int1;
typedef unsigned char  TD_uint1;
typedef short          TD_int2;
typedef unsigned short TD_uint2;
typedef int            TD_int;
typedef int            TD_int4;
typedef unsigned int   TD_uint4;
#endif
```

2.1.3 FUNCTIONS PROVIDED

The PS700Driver C is divided into the callable functions in Table 2-1. These functions are similar to the subcommands used in the PIC® device-based driver:

- `TD_int1 p7fgc_proc()`
- `TD_int1 p7fgc_reset()`
- `TD_int1 p7fgc_init()`
- `TD_int1 p7fgc_capset(TD_int2 cap_mah)`
- `TD_int1 p7fgc_version(TD_uint2 *ver)`

TABLE 2-1: PS700DRIVER C FUNCTIONS

Type	Name	Definition
`TD_int1`	`p7fgc_proc()`	**Return Value:** Function returns 0 = no error, !0 = error
		Description: Performs fuel gauge processing sequence communicating with the PS700 as needed
`TD_int1`	`p7fgc_reset()`	**Return Value:** Function returns 0 = no error, !0 = error
		Description: Ensures the fuel gauge is in a “coherent” state after system Reset
`TD_int1`	`p7fgc_init()`	**Return Value:** Function returns 0 = no error, !0 = error
		Description: Executed once to upload static parameters from a newly installed battery
`TD_int1`	`p7fgc_capset(TD_int2 cap_mah)`	**Return Value:** Function returns 0 = no error, !0 = error
		Description: Executed to set the fuel gauge to an arbitrary capacity value
		Arguments: “`cap_mah`” – desired capacity value in units of mAh
`TD_int1`	`p7fgc_version(TD_uint2 *ver)`	**Return Value:** Function returns 0 = no error, !0 = error
		Description: Return version number
		Arguments: “`ver`” – pointer to storage for version number

2.1.4 ADDITIONAL FUNCTIONS

Since communication between the host and the PS700 is system dependent, the following SMBus communication functions must be written by the integrator:

- `TD_int1 p7fgc_p7_read(TD_int2 p7_buffer, TD_int1 *ram_buffer, TD_int1 nbytes)`
- `TD_int1 p7fgc_p7_write(TD_int1 *ram_buffer, TD_int2 p7_buffer, TD_int1 nbytes)`

TABLE 2-2: SMBus COMMUNICATION FUNCTIONS

Type	Name	Definition
`TD_int1`	`p7fgc_p7_read`	**Return Value:** Function returns 0 = no error, !0 = error
		Description: Host API routine used to read data from the PS700
		Arguments: "`p7_buffer`": PS700 memory address – source data "`ram_buffer`": destination address "`nbytes`": number of bytes to transfer
`TD_int1`	`p7fgc_p7_write`	**Return Value:** Function returns 0 = no error, !0 = error
		Description: Host API routine used to write data to the PS700
		Arguments: "`ram_buffer`": memory address – source data "`p7_buffer`": destination address in PS700 "`nbytes`": number of bytes to transfer

Note: These functions refer to PS700 memory addresses as 16-bit values (i.e., 0000bbaa aaaaaaaa – defines 10-bit address "aaaaaaaaaa" in bank "bb").

Each routine (above) is required to set global variable flag "`g_p7fgc_cerr = 1`" to indicate a communication error of any sort. This flag is cleared externally to these routines.

2.2 Math

All calculations are performed in integer math. In lieu of the explicit math firmware functions in the PS700Driver, the PS700Driver C compiler is allowed to automatically link library routines for math operations (+, -, *, /).

2.3 Data

Global variables, used for the variables and registers in the PS700Driver, are defined to match the size and type. Global variable names are prefaced with "`g_p7fg_`". The full suite of variables is defined in `p7fgc.h`.

2.4 Memory Map

The memory locations of the PS700 hardware registers are defined in `p7fgc_dev.h`.

3.0 EXCEPTIONS

The PS700Driver C is intended to be the "C" version of the PS700Driver for the PIC microcontroller. The design of the PS700Driver was dictated in some cases by the limited resources of the PIC microcontroller. Those design compromises were carried forward into the PS700Driver C, not out of necessity, but to maintain parity between the two. In other areas, the PS700Driver C deviated slightly from the PS700Driver for PIC microcontrollers for portability, comprehending the reality of various and unknown host platforms. Features in each category are discussed individually below.

3.1 PS700Driver C Differences

3.1.1 DATA TRANSFER TO/FROM PS700

The PS700Driver C makes no assumptions concerning RAM data storage and/or alignment. All "blocks" of data transferred to/from the PS700 are moved one multi-byte variable at a time, or a block is transferred into a temporary byte array and each component variable is loaded individually. The PS700Driver can transfer a block of data to/from the PS700 and directly overlay variables in the microcontroller RAM. See Table 3-1 for the affected modules.

TABLE 3-1: DATA TRANSFER MODULES

Function	Description
`p7fgc_sync_0()`	Read/upload static variables from PS700 EEPROM to fuel gauge RAM
`p7fgc_sync_1()`	Read/upload fuel gauge context from PS700 RAM to fuel gauge RAM
`p7fgc_dsync_1()`	Write/download fuel gauge context from fuel gauge RAM to PS700 RAM

3.2 PS700Driver C Continuance of PS700Driver Compromises

The following features were voluntarily maintained as implemented in the PS700Driver, which has limited PIC architecture resources. Additional resources available to the PS700Driver C may create opportunities to revise these features, as needed, to improve performance.

3.2.1 PARTIAL STORAGE OF VEOD LUT

To minimize communication with the PS700 and RAM usage, only one row of the 2D VEOD LUT, corresponding to the current temperature, is stored locally and used until the temperature changes. Routine communication could be eliminated totally by uploading the entire 4 by 8 table into RAM during initialization.

3.2.2 OPERATIONAL FLAGS

The PS700Driver C uses 1-byte variables set to '`0`' or '`1`' to implement the operational "bit" flags in the PS700Driver.

3.2.3 SELF-DISCHARGE

The self-discharge algorithm was implemented by equation. A self-discharge Look-up Table (LUT) has been historically used and may be preferred.

3.2.4 NON-INTERPOLATED RESIDUAL CAPACITY

To minimize code space and execution time, the residual capacity LUT was implemented with simple binning; interpolation may be preferred. To mitigate this compromise, the temperature axis is defined specifically for residual capacity independent of the temperature axis for the VEOD LUT.

3.2.5 MATH

Math operations in the PS700Driver are staged to accommodate integer math and a maximum 16x16 multiply and a 32/16 divide. If host math is not constrained as such, two issues can be eliminated or modified:

- Scaling operands in `g_p7fgc_proc_cap` to fit 16x16 multiplies
- Piecemeal implementation of the self-discharge equation to fit 16x16 multiplies (`p7fgc_sdchg()`)

4.0 IMPLEMENTATION

The remainder of this document describes the functionality of the PS700Driver for PIC micro-controllers. The differences between the PIC assembly code version (PS700Driver) and the “C” code version (PS700Driver C) have been highlighted in **Section 3.0 “Exceptions”**.

The PS700Driver, as implemented on the Microchip PowerInfo™ 2 configuration interface, is used as an example. The PowerInfo 2 (PS051) is available in the PS7070EV evaluation kit for the PS700. As implemented on the PowerInfo 2, the PS700Driver is allocated the following resources:

- Program memory – page 3 (addresses x1800-x1FFF)
- RAM – Bank 3 (addresses x190-x1EF)

Because the PowerInfo 2 operates as a slave to the remote host master, two custom commands were added to the PowerInfo 2 repertoire for remote control of the PS700Driver:

1. Read/write PowerInfo 2 RAM (previously, only Bank 1 read/write was available).
2. Execute the PS700Driver.

Executing the PS700Driver is a two step process. First, write the subcommand to RAM and then call the PS700Driver.

The PS700Driver communication needs are supplied by the standard PowerInfo 2 SMBus state controller. Because this controller is “script-driven” internally, it can implement arbitrary protocol(s). The custom SMBus API needed for the PS700Driver merely generates the script for PS700 SMBus protocol (based on the desired addresses and byte count). Control is then passed to the PowerInfo 2 SMBus module to execute the transaction.

4.1 Application Requirements

The PS700Driver requires calls from the application firmware at regular intervals (see **Section 4.2 “Execution Model”**) and callable SMBus I/O routine(s) for communication with the battery (see **Section 4.3 “SMBus Communication”**).

4.2 Execution Model

The PS700Driver is called by the host firmware as a routine task. The interval between calls can vary in length from seconds to hours but it’s efficacy depends on conditions. A Fuel Gauge (FG) status bit, “TURBO”, can be used as an indication to shorten the interval if the fuel gauge is close to a time critical trigger point, such as End-Of-Charge (EOC) or End-Of-Discharge (EOD). The PS700Driver has a single entry point with multiple subfunctions. The subfunction command code is written to the memory scratchpad before the driver is called. Currently, four subfunctions are defined:

1. Reset – Executed once when the host firmware itself is initialized. Reset ensures the fuel gauge is in a coherent state after system start-up.
2. Initialization – Executed once to upload static parameters from a newly installed battery. Initialization could be automatically executed by FG processing but a separate subcommand gives the host more latitude in scheduling this process, which requires additional (non-routine) communication with the battery.
3. Capacity Set – Executed to force the fuel gauge to register an arbitrary capacity value.
4. Fuel Gauging – Executed at a routine interval; reads then processes dynamic data from the PS700. Results are stored in RAM.

4.3 SMBus Communication

The PS700Driver requires SMBus I/O for communication with the PS700. Because this functionality is highly system dependent, it is not part of the PS700Driver. The application must provide callable modules to implement the PS700 SMBus protocol for reading and writing.

The PS700Driver provides a device memory address (PS700), local memory address (RAM) and a byte count to either a “read block” or “write block” routine. Upon return from the communication routine(s), a failure is indicated by FLAG_P7FG_COMM_ERR = `1`. No action is taken on FLAG_P7FG_COMM_ERR if the transaction is successful.

If the application’s SMBus routines cannot be altered, custom SMBus API firmware must provide an interface which supports the PS700 SMBus protocol between the application and the PS700Driver.

5.0 HOST

5.1 PS700Driver Interface

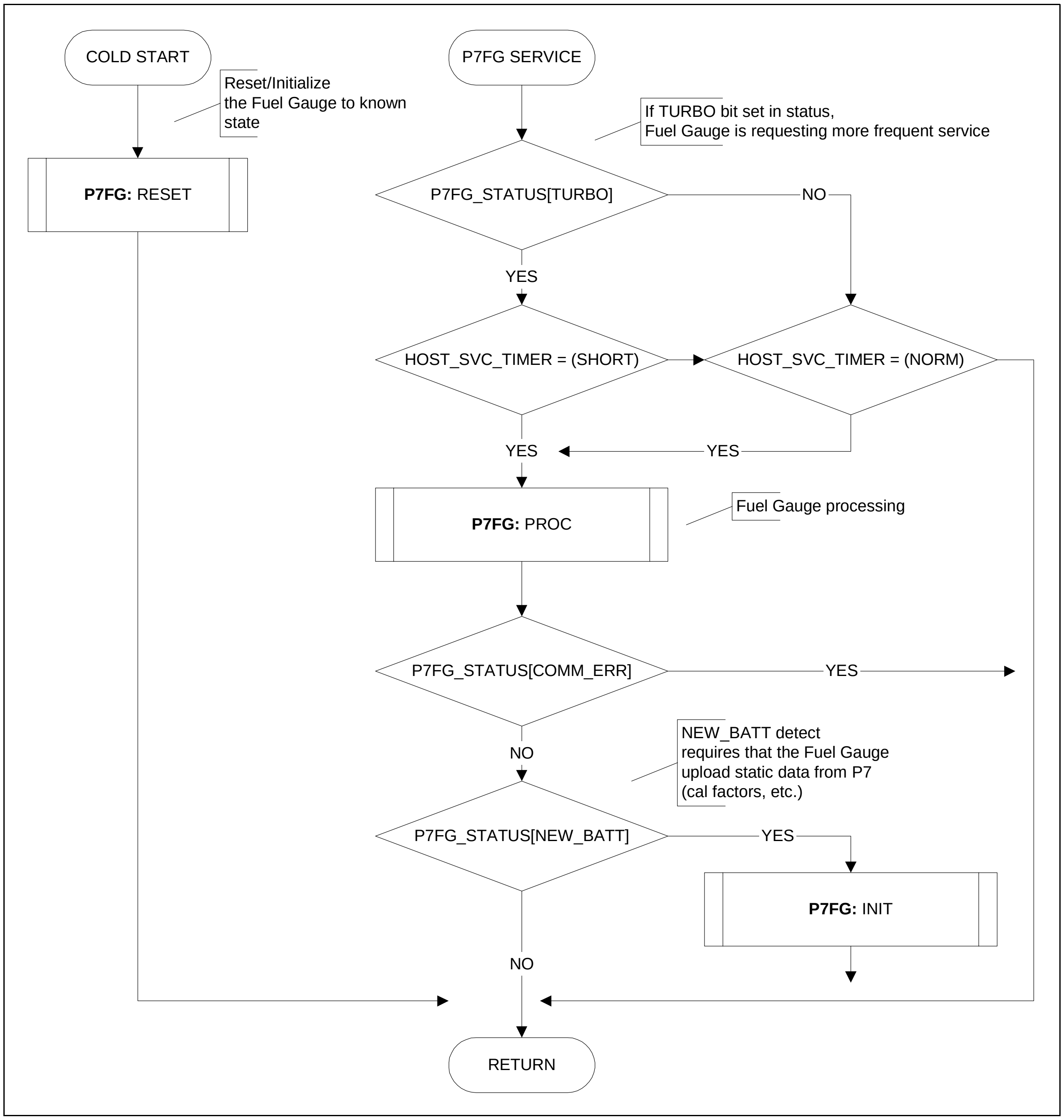
As described above, the PS700Driver is a modular “add on” to the application firmware. The application and the PS700Driver interface in only three ways:

1. Host calls the PS700Driver for execution.
2. The PS700Driver calls the host for SMBus facilities.
3. Common RAM provides a passive data interface.

5.1.1 EXECUTION

The host is expected to call the PS700Driver on a routine basis. To minimize the burden on the host, the interval between calls can vary from seconds to hours. The PS700Driver status (TURBO) bit can indicate the need to shorten the interval during time critical junctures. A nominal service interval could be 1 minute (normal) and 10 seconds (critical). When called, the PS700Driver expects to complete execution without significant interruption. The host execution flow chart appears in Figure 5-1.

FIGURE 5-1: HOST EXECUTION FLOW CHART


COLD START
Reset/Initialize the Fuel Gauge to known state
P7FG: RESET
P7FG SERVICE
If TURBO bit set in status, Fuel Gauge is requesting more frequent service
P7FG_STATUS[TURBO]
NO
YES
HOST_SVC_TIMER = (SHORT)
HOST_SVC_TIMER = (NORM)
YES
YES
P7FG: PROC
Fuel Gauge processing
P7FG_STATUS[COMM_ERR]
YES
NO
NEW_BATT detect requires that the Fuel Gauge upload static data from P7 (cal factors, etc.)
P7FG_STATUS[NEW_BATT]
YES
P7FG: INIT
NO
RETURN

5.1.2 SMBus COMMUNICATION

During execution, the PS700Driver must rely on host facilities for SMBUS transactions. Because RAM space is limited, data is read from the PS700 one register at a time as needed. To simplify, all communication is done using Block mode; therefore, only two constructs are needed: (1) read block and (2) write block. A transaction must have three pieces of data: (1) device memory address, (2) local RAM memory address and (3) byte count. In the PS700Driver, this information is available in three temporary registers.

5.1.3 DATA

All data/parameters required by the PS700Driver are kept in RAM. Static parameters read from the PS700 are stored in RAM. Results of fuel gauge processing are also stored in system RAM. The data interface between the host and the PS700Driver is the common memory model.

5.2 Resources

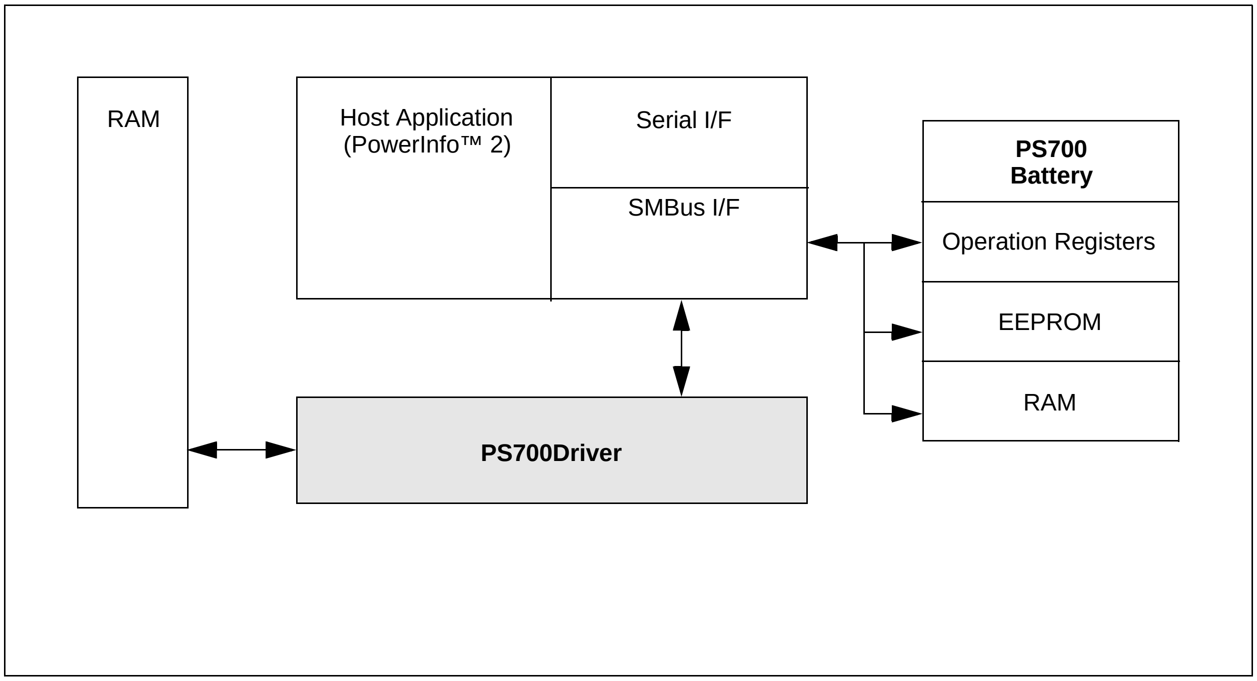
The host must allocate resources (a block of program memory and a block of RAM) to the PS700Driver. Unaltered, the PS700Driver requires a maximum of 2k words of program memory and 96 bytes of RAM. Any changes to the PS700Driver may also change the resources required.

6.0 DRIVER

6.1 Data

The PS700Driver data is located both remotely in the battery and locally in system RAM (see Figure 6-1).

FIGURE 6-1: PowerInfo™ 2 ARCHITECTURE – DATA STORAGE



6.1.1 SYSTEM RAM

All static data (parameters), dynamic data (results of processing) and scratchpad used by PS700Driver and available to the host is maintained in system RAM.

6.1.2 PS700 OPERATION REGISTERS

All data for PS700 hardware control is located in the operation registers. Please see the *"PS700 Data Sheet"* (DS21760) for detailed register information.

6.1.3 PS700 EEPROM

All critical PS700 parameters, calibration factors and learned data are stored in PS700 integrated EEPROM. See Table 6-1 for the PS700 EEPROM map.

TABLE 6-1: PS700 EEPROM MAP

Address[1]	Name	LEN	Units	Description
0x00	CF_CURR	2	N/A	Calibration Factor – Gain - Current
0x02	CO_CURR	2	AD	Calibration Factor – Offset – Current
0x04	CF_PACK	2	N/A	Calibration Factor – Gain – Pack Voltage
0x06	CF_VCELL1	2	N/A	Calibration Factor – Gain – Cell 1 Voltage
0x08	CF_VCELL2	2	N/A	Calibration Factor – Gain – Cell 2 Voltage
0x0A	CF_TEMP	2	N/A	Calibration Factor – Gain – Temperature
0x0C	CO_TEMP	2	N/A	Calibration Factor – Offset – Temperature
0x0E	VEOD	2	mV	EOD Voltage Threshold
0x10	VEOC	2	mV	EOC Voltage Threshold
0x12	IEOC	2	mA	EOC Current Threshold
0x14	EOD_CAP	2	mAh	EOD Capacity
0x16	MODE	1	bits	Fuel Gauge Mode Bits
0x17	SERIAL_NO	2	N/A	Serial Number – Battery ID
0x19	CAP_FULL	2	mAh	Full Charge Capacity
0x1B	CYCLES	2	cycles	Cycle Count
0x1D	VEOD_C	3	XmA	VEOD LUT Current Axis
0x20	VEOD_T	7	XdegC	VEOD LUT Temperature Axis
0x27	VEOD	32	XmV	VEOD Voltage Threshold(s)
0x47	RCAP_T	7	XdegC	RCAP LUT – Residual Capacity Temperature Axis
0x4E	RCAP	8	XmAh	Residual Capacity
	TOTAL	**86**		

Note 1: Address in table is relative to origin 0xA0.

6.1.4 PS700 RAM

The PS700Driver stores fuel gauge context in PS700 RAM. The context is defined as the essential fuel gauge data (capacity and time) used as the basis for operation. This data is written to the PS700 RAM following each processing sequence. This information is read from PS700 RAM whenever a processing sequence is interrupted. In the case of a removable PS700 battery pack, an interruption can occur as a result of a pack change. Interruptions also occur if errors are detected in communication. See Table 6-2 for the PS700 RAM map.

TABLE 6-2: PS700 RAM MAP

Address[1]	Name	LEN	Units	Description
0x00	CAP_CYC	2	mAh	Cycle Counting – Capacity Accumulation
0x02	CAP_LAST	2	mAh	Cycle Counting – Last Value of Capacity
0x04	CAP_UNITS	4	500 mSmA	Fuel Gauge Capacity from Last PROC Sequence
0x08	TIME_LAST	4	500 mS	Device Time from Last PROC Sequence
0x0C	CAP_LAST_D	4	500 mSAD	Device Capacity from Last PROC Sequence
0x10	CHECKSUM	1	N/A	Checksum = (sum of bytes 0x00-0x0f) + K,K = 0x12 (arbitrary bias)
	TOTAL	**17**		

Note 1: Address in table is relative to origin 0x00.

6.1.5 SYSTEM RAM

The PS700Driver stores fuel gauging results in system RAM to limit the need to communicate with the PS700. Static variables from the PS700 EEPROM are also stored in system RAM for this reason. See Table 6-3 for the system RAM map.

TABLE 6-3: SYSTEM RAM MAP

Address(1)	Name	LEN	Units	Description
0x00	CURR	2	mA	Current – Processed ADC Result
0x02	CURR_AVG	2	mA	Average Current – Capacity-based Calculation
0x04	VPACK	2	mV	Pack Voltage – Processed ADC Result
0x06	VCELL1	2	mV	Cell 1 Voltage – Processed ADC Result
0x08	VCELL2	2	mV	Cell 2 Voltage – Processed ADC Result
0x0A	TEMP	2	degC	Temperature – Processed ADC Result
0x0C	CAP	2	mAh	Capacity
0x0E	CAP_REL	1	%	Relative Capacity – % of CAP_FULL
0x0F	STATUS	2	bits	Fuel Gauge Status – see Table 6-4
0x11	CAP_RES	2	mAh	Residual Capacity
0x13	CAP_CYC	2	mAh	Capacity Accumulation – Cycle Counting
0x15	CAP_LAST	2	mAh	Last Value of CAP – Cycle Counting
0x17	CAP_UNITS	4	500 mSmA	Capacity
0x1B	TIME_LAST	4	500 mS	Device Tie from Last PROC Sequence
0x1F	CAP_LAST_D	4	500 mSAD	Device Capacity from Last PROC Sequence
0x23	CHECKSUM	1	N/A	Checksum of Fuel Gauge Context
0x24	CF_CURR	2	N/A	Calibration Factor – Gain – Current
0x26	CO_CURR	2	AD	Calibration Factor – Offset – Current
0x28	CF_PACK	2	N/A	Calibration Factor – Gain – Pack Voltage
0x2A	CF_VCELL1	2	N/A	Calibration Factor – Gain – Cell 1 Voltage
0x2C	CF_VCELL2	2	N/A	Calibration Factor – Gain – Cell 2 Voltage
0x2E	CF_TEMP	2	N/A	Calibration Factor – Gain – Temperature
0x30	CO_TEMP	2	N/A	Calibration Factor – Offset – Temperature
0x32	VEOD	2	mV	EOD Voltage Threshold
0x34	VEOC	2	mV	EOC Voltage Threshold
0x36	IEOC	2	mA	EOC Current Threshold
0x38	EOD_CAP	2	mAh	EOD Capacity
0x3A	MODE	1	bits	Fuel Gauge Mode Bits – see Table 6-5
0x3B	SERIAL_NO	2	N/A	Serial Number – Battery ID
0x3D	CAP_FULL	2	mAh	Full Charge Capacity
0x3F	CYCLES	2	cycles	Cycle Count
0x41	VEOD_C1	1	mAx (1)	VEOD LUT Current Axis Point
0x42	VEOD_C2	1	mAx (1)	VEOD LUT Current Axis Point
0x43	VEOD_C3	1	mAx (1)	VEOD LUT Current Axis Point
0x44	VEOD_V1	1	mVx (2)	VEOD LUT Voltage Vector (row in table)
0x45	VEOD_V2	1	mVx (2)	VEOD LUT Voltage Vector (row in table)

Note 1: Address in table is relative to origin 0x190.

TABLE 6-3: SYSTEM RAM MAP (CONTINUED)

Address(1)	Name	LEN	Units	Description
0x46	VEOD_V3	1	mVx (2)	VEOD LUT Voltage Vector (row in table)
0x47	VEOD_V4	1	mVx (2)	VEOD LUT Voltage Vector (row in table)
0x48	T_SAVE	1	degC	Temperature Compare Save Register
0x49	SCRATCHPAD	14	N/A	Seven 16-bit Accumulators, Command Buffer, etc.
0x57	(INTERNAL)	6	N/A	Miscellaneous Internal Variables
0x5D	COMM_ADR	1	N/A	Communications Module, Local Ram Address
0x5E	COMM_ADR_D	1	N/A	Communications Module, Device Address (MSB)
0x5F	Not Used	1	N/A	Not Used
	TOTAL	**96**		

Note 1: Address in table is relative to origin 0x190.

TABLE 6-4: STATUS MAP

Bit	Name	Description
15	TURBO	= 1, Near trigger point(s), request frequent processing interval(s)
14	Not Assigned	
13	Not Assigned	
12	Not Assigned	
11	Not Assigned	
10	Not Assigned	
09	EOC	= 1, EOC (End-Of-Charge) detect
08	EOD	= 1, EOD (End-Of-Discharge) detect
07	Not Assigned	
06	DISCHARGING	Fuel gauge measuring decreasing capacity (discharging)
05	FULL_CHG	Fully charged, set at EOC, reset at predetermined RSOC below EOC
04	FULL_DCHG	Fully discharged, set at EOD, reset at predetermined RSOC above EOD
03	WARM_START	Due to issue(s) during last processing cycle, fuel gauge read “context” (starting point) from PS700 RAM
02	DEFAULT_CONTEXT	Fuel gauge read context from battery (PS700 RAM), data was determined to be suspect (bad checksum), default context set fuel gauge will most likely be in error until EOD or EOC is encountered
01	NEW_BATTERY	Fuel gauge detected “new” battery (i.e., serial number different), the host must issue an “initialization” command before any processing can continue
00	COMM_ERR	Communication error with battery on the next processing request, the fuel gauge will attempt to restart by reading the previous context from the battery

TABLE 6-5: MODE MAP

Bit	Name	Description
7	BN_MODE_1CELL	= 1: one cell, = 0: two cell
6	Not Assigned	
5	Not Assigned	
4	Not Assigned	
3	BN_MODE_VEOD_K	= 1: constant VEOD, = 0: used VEOD LUT
2	BN_MODE_RCC_DIS	= 1: REMCAP compensation disabled, = 0: enabled
1	BN_MODE_SDCHG_DIS	= 1: self-discharge feature disabled, = 0: enabled
0	BN_MODE_ADC_DIS	= 1: disable ADC processing (low-level diag.), = 0: normal operation

7.0 PROCESSING

The PS700Driver is designed as a single thread, single entry point callable module. Calling the PS700Driver is a 2-step process:

1. Write subcommand code to RAM.
2. Execute PS700Driver by calling the entry point.

The main process is detailed in Figure 7-1. See Table 7-1 for a description of the processing commands.

FIGURE 7-1: MAIN PROCESSING FLOW CHART

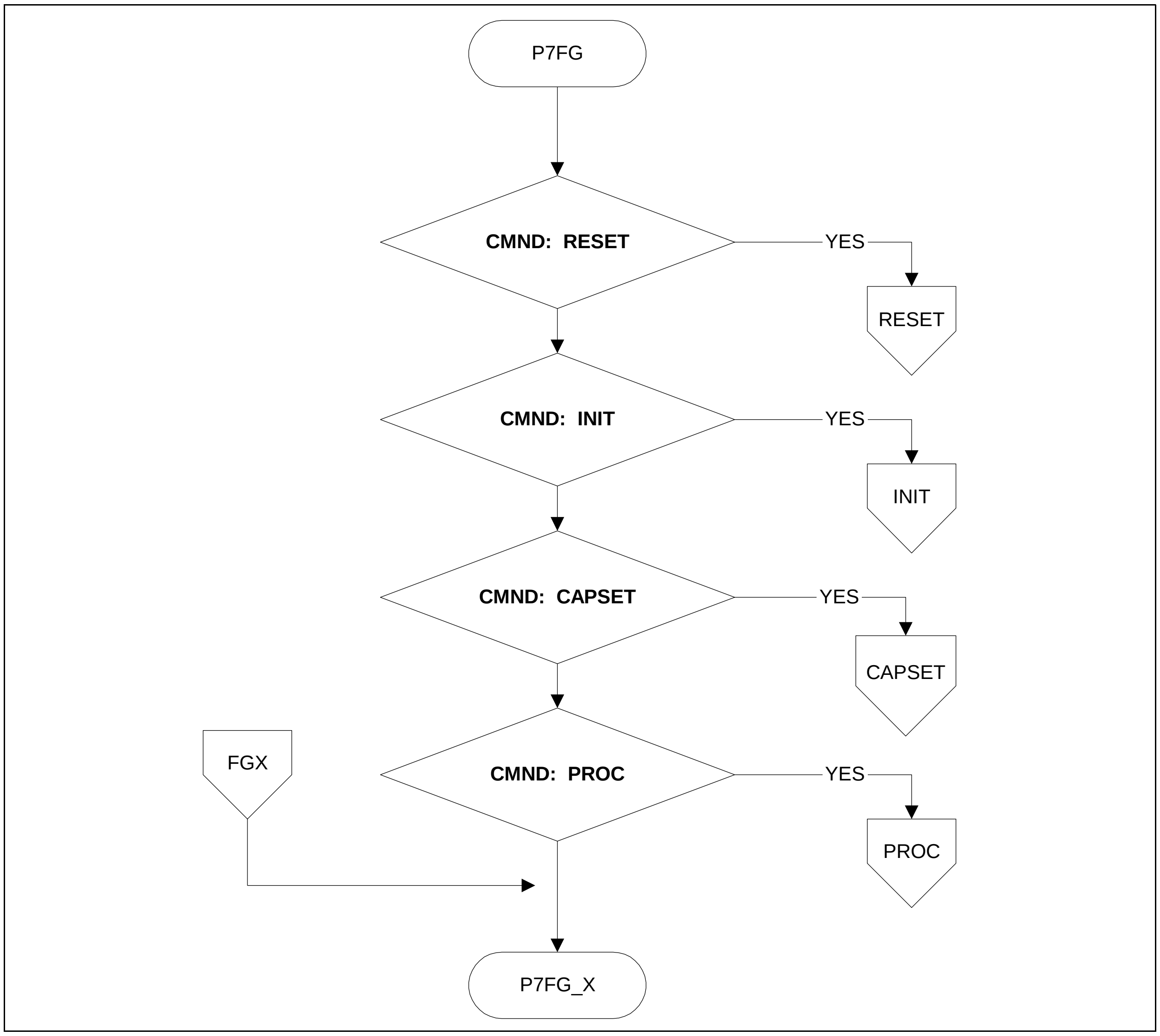


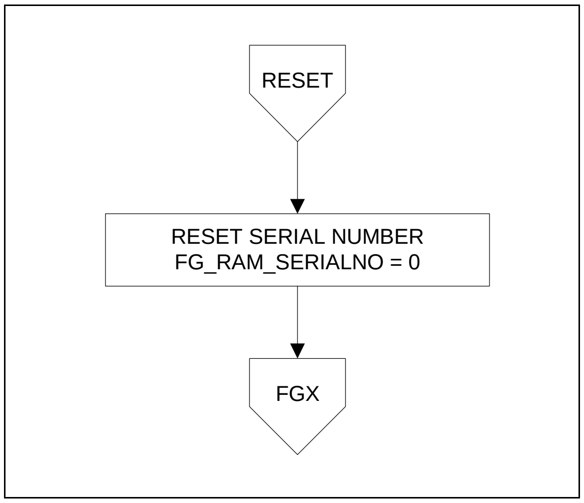
TABLE 7-1: PROCESSING COMMANDS

CMND#	Name	Description
1	RESET	Executed once when the host firmware itself is initialized – insures the fuel gauge is in a coherent state.
2	INIT	Executed once to upload static parameters from a newly installed battery.
3	CAPSET	Executed to force the fuel gauge to register an arbitrary capacity value.
4	PROC	Executed at a routine interval – reads then processes dynamic data from the PS700 battery. Results are stored in RAM.

7.1 Subcommand RESET

The RESET subcommand is used to insure that the fuel gauge is properly initialized after a system Reset. The host need only call RESET and the PS700Driver will perform the necessary housekeeping (e.g., zero dynamic flags, clear battery ID to force “new battery” processing, etc.). The RESET process is detailed in Figure 7-2.

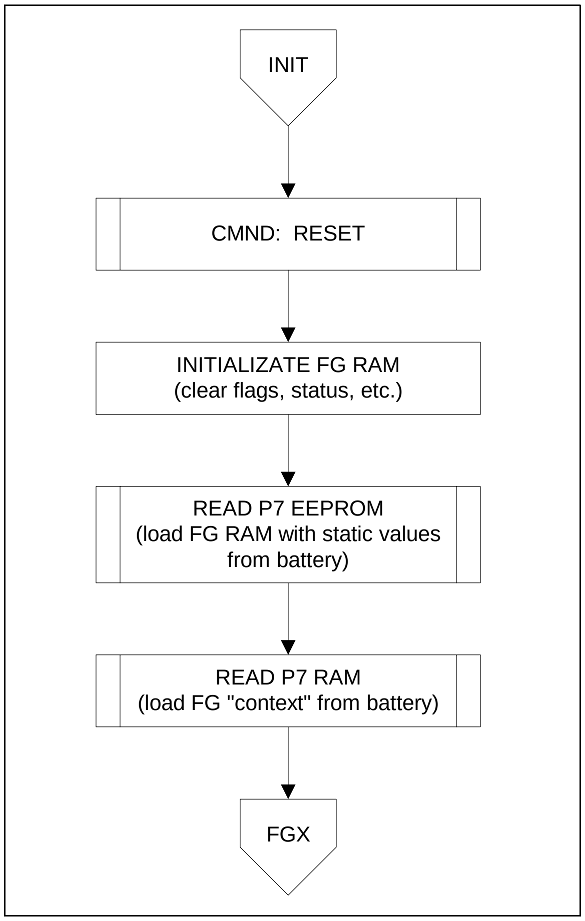
FIGURE 7-2: RESET FLOW CHART



7.2 Subcommand INIT

Subcommand INIT directs the PS700Driver to upload static data from the battery. INIT is executed when the PS700Driver senses a new battery. Static data is read once from the battery and saved in system RAM, reducing as much as possible the need to communicate with the battery. INIT could be executed automatically by the PS700Driver, but a separate subcommand gives the host more latitude in scheduling this process, which requires additional (non-routine) communication with the battery. The INIT process is detailed in Figure 7-3.

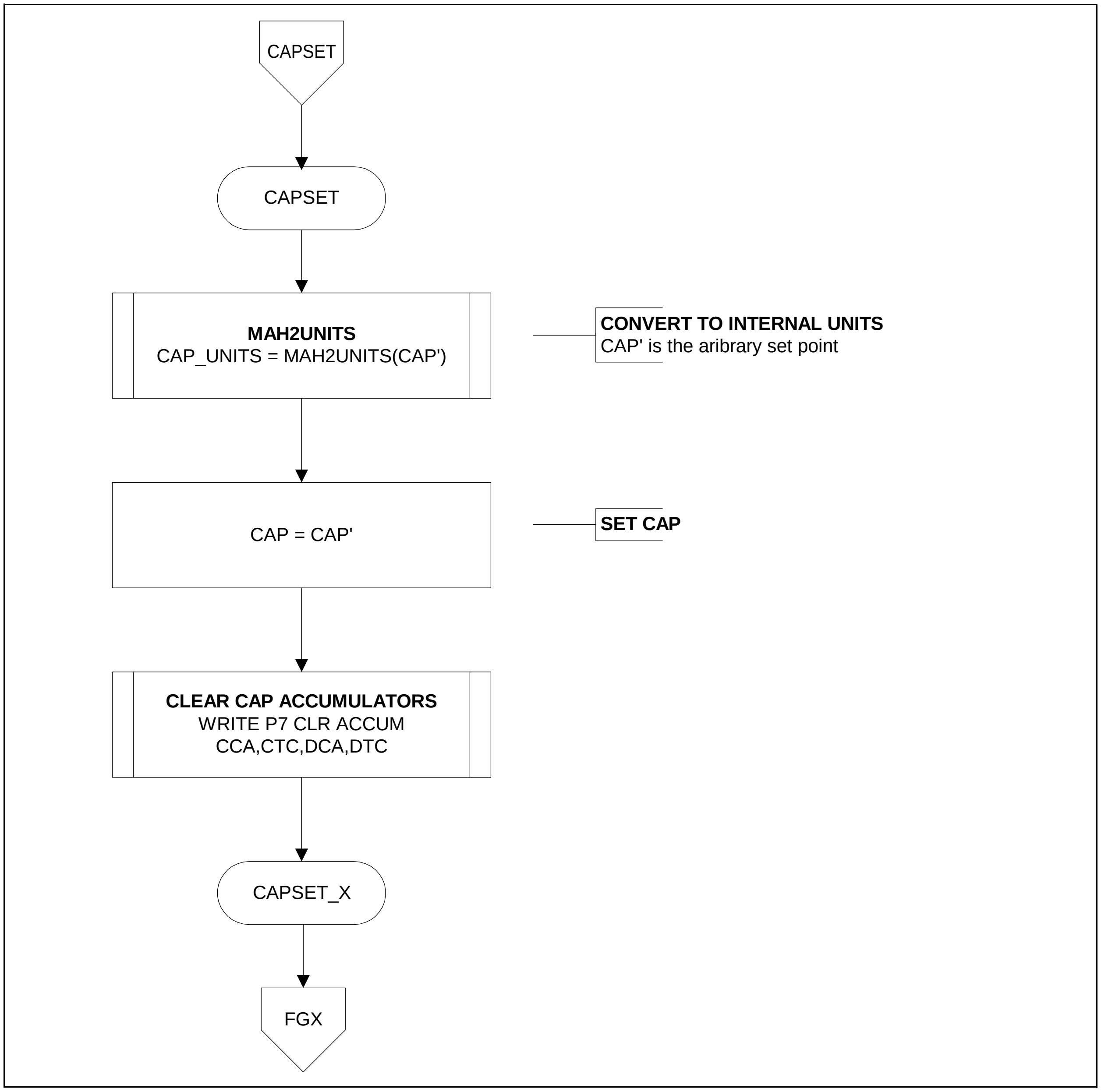
FIGURE 7-3: INIT FLOW CHART


INIT
CMND: RESET
INITIALIZATE FG RAM
(clear flags, status, etc.)
READ P7 EEPROM
(load FG RAM with static values from battery)
READ P7 RAM
(load FG "context" from battery)
FGX

7.3 Subcommand CAPSET

CAPSET allows the host to set the remaining capacity initially reported by the fuel gauge. As seen in the flow chart, it is a multi-step process. The PS700Driver will load the value for capacity (converting it to the proper units) and reset the PS700 accumulators. The fuel gauge context will then be written to PS700 RAM.

FIGURE 7-4: CAPSET FLOW CHART



7.4 Subcommand PROC

The PROC command performs the fuel gauging, communicating with the PS700 as required. The results of processing are written to system RAM. PROC is detailed in Figure 7-5 and Figure 7-6. A description of each block in PROC is in Table 7-2.

FIGURE 7-5: PROC FLOW CHART (PART 1)

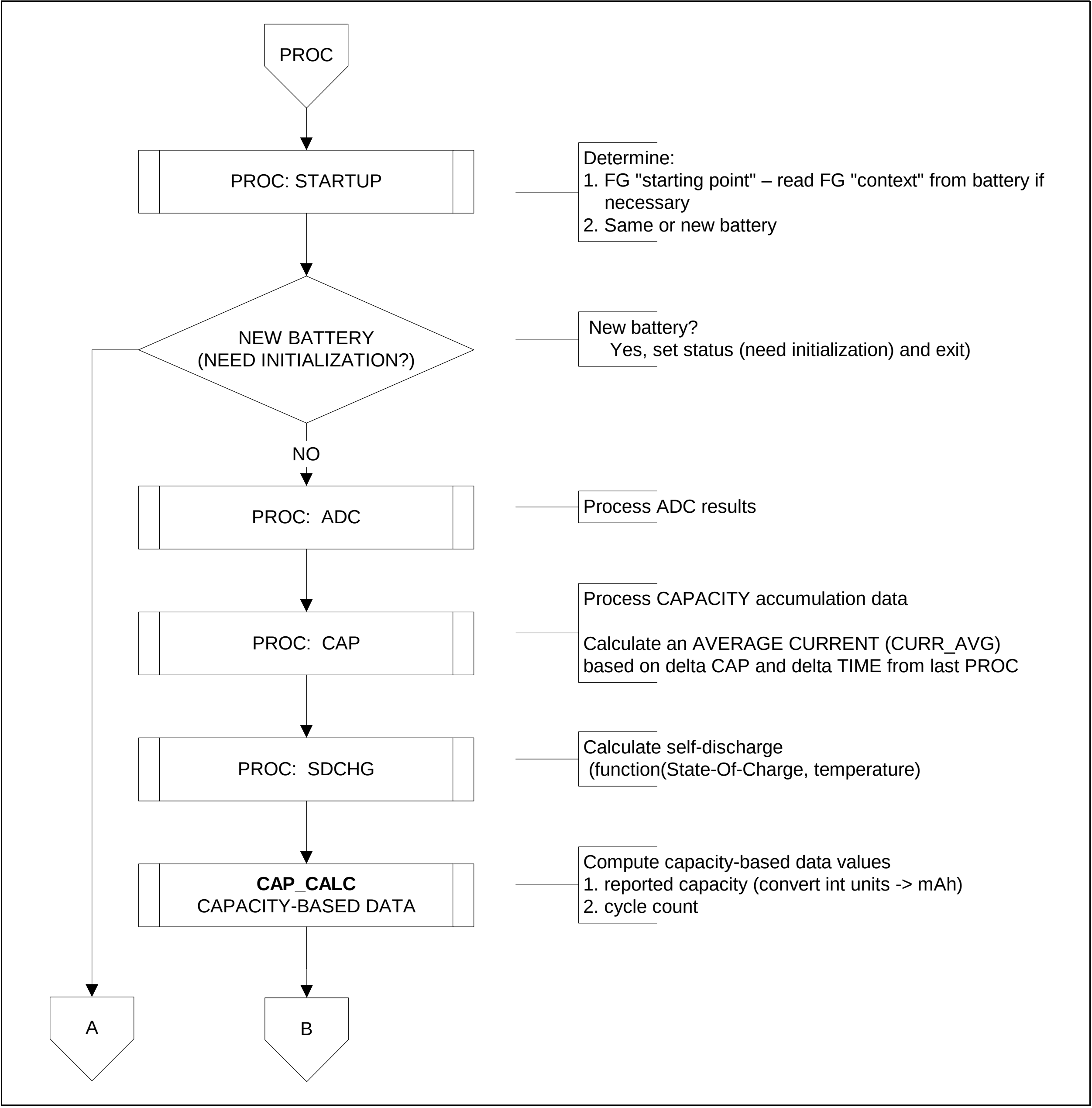

PROC
PROC: STARTUP
Determine:
1. FG "starting point" – read FG "context" from battery if necessary
2. Same or new battery
NEW BATTERY (NEED INITIALIZATION?)
New battery?
Yes, set status (need initialization) and exit)
NO
PROC: ADC
Process ADC results
PROC: CAP
Process CAPACITY accumulation data
Calculate an AVERAGE CURRENT (CURR_AVG) based on delta CAP and delta TIME from last PROC
PROC: SDCHG
Calculate self-discharge (function(State-Of-Charge, temperature)
CAP_CALC
CAPACITY-BASED DATA
Compute capacity-based data values
1. reported capacity (convert int units -> mAh)
2. cycle count
A
B

FIGURE 7-6: PROC FLOW CHART (PART 2)

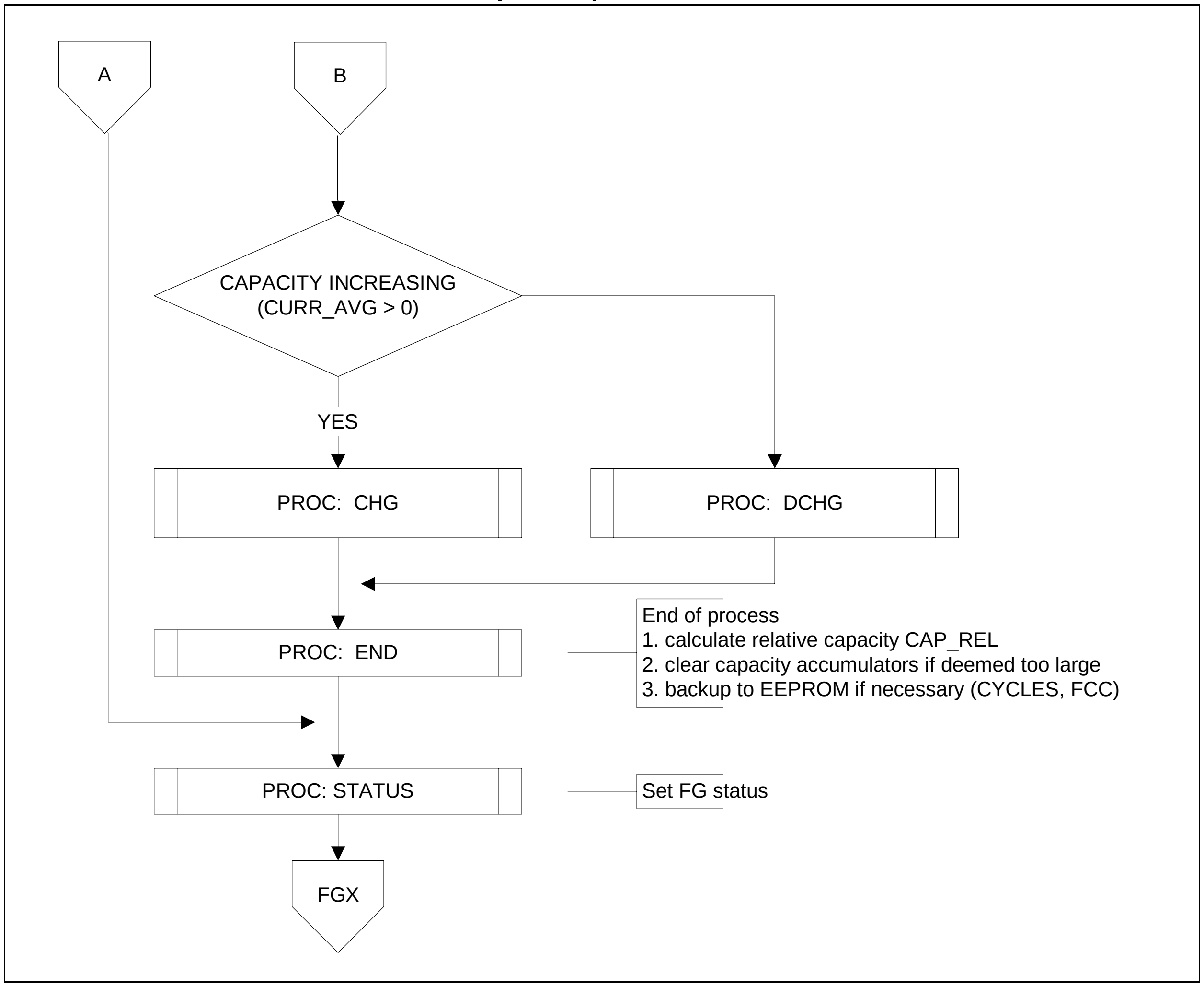

A
B
CAPACITY INCREASING
(CURR_AVG > 0)
YES
PROC: CHG
PROC: DCHG
PROC: END
End of process
1. calculate relative capacity CAP_REL
2. clear capacity accumulators if deemed too large
3. backup to EEPROM if necessary (CYCLES, FCC)
PROC: STATUS
Set FG status
FGX

TABLE 7-2: FUEL GAUGING (PROC) BLOCKS

Step	Name	Description	PS700 Data I/O
1	START-UP	Read fuel gauge “context” from PS700 RAM (if necessary)	R 17 bytes
2	ADC PROC	Read/process ADC readings	R 10 bytes
3	CAP PROC	Read/process capacity accumulators	R 16 bytes
4	SELF DCHG	Read/process timer, if timer ready – compute self-discharge Reset timer	R 4 bytes W 1 byte
5	CAP CALC	Calculate capacity-based derivative data	0 bytes
6	STATE	Charge or discharge processing	0 bytes
7	END	Build status, reset accumulators (if necessary), write fuel gauge “context” to PS700 RAM	0 bytes W 1 byte W 17 bytes

7.4.1 BLOCK START-UP

The start-up phase first checks communication with the battery. If communication is possible, the PS700Driver reads the serial number. If the serial number is different from what had been read previously, the “new battery” status bit is set and control is returned to the host. If the serial number is the same, processing continues. If a communication error occurred during the last call to PROC, the fuel gauge context is read from the PS700 RAM to start fuel gauge calculations from a known state. Figure 7-7 and Figure 7-8 show details of the start-up block.

FIGURE 7-7: START-UP FLOW CHART (PART 1)

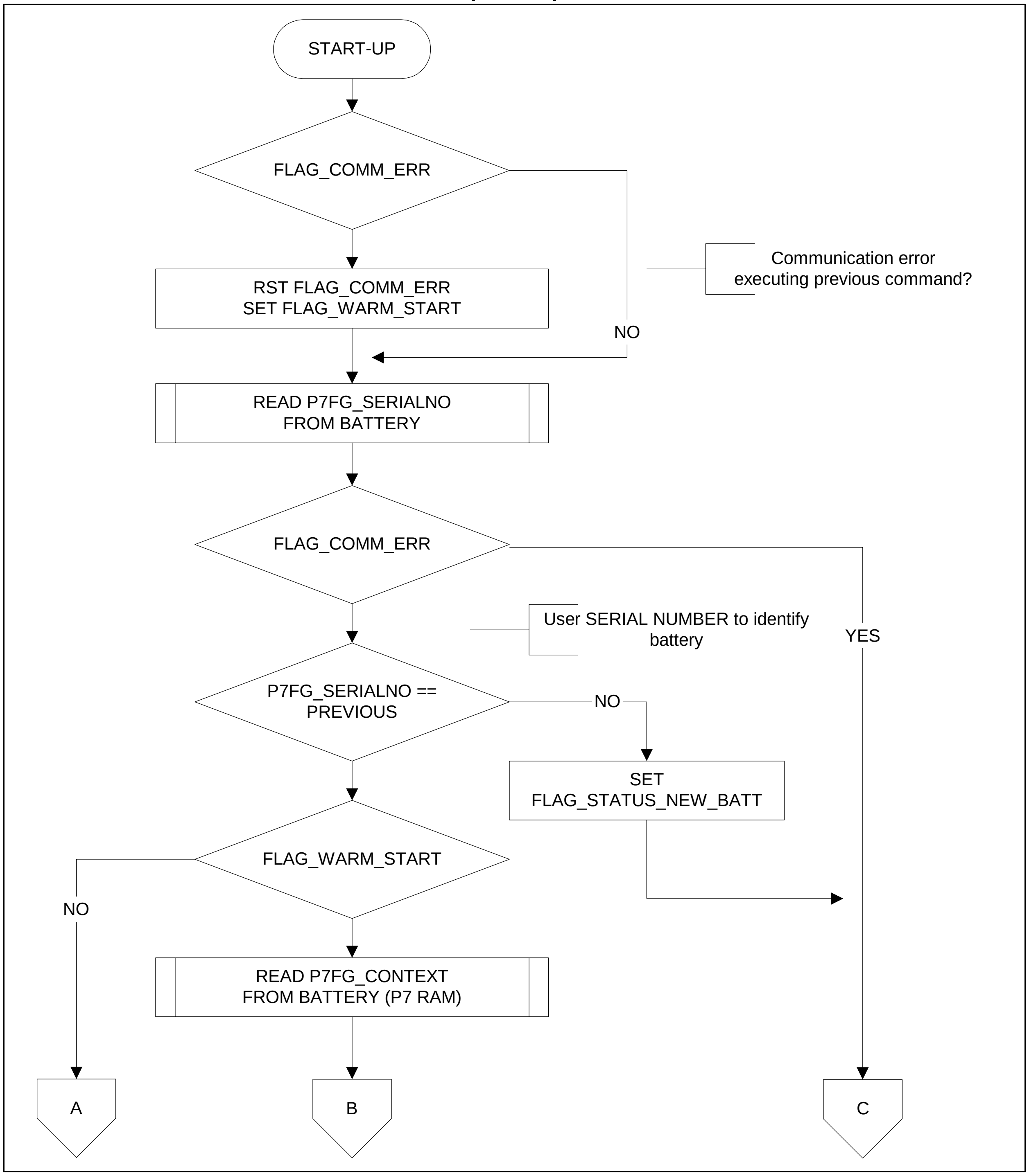
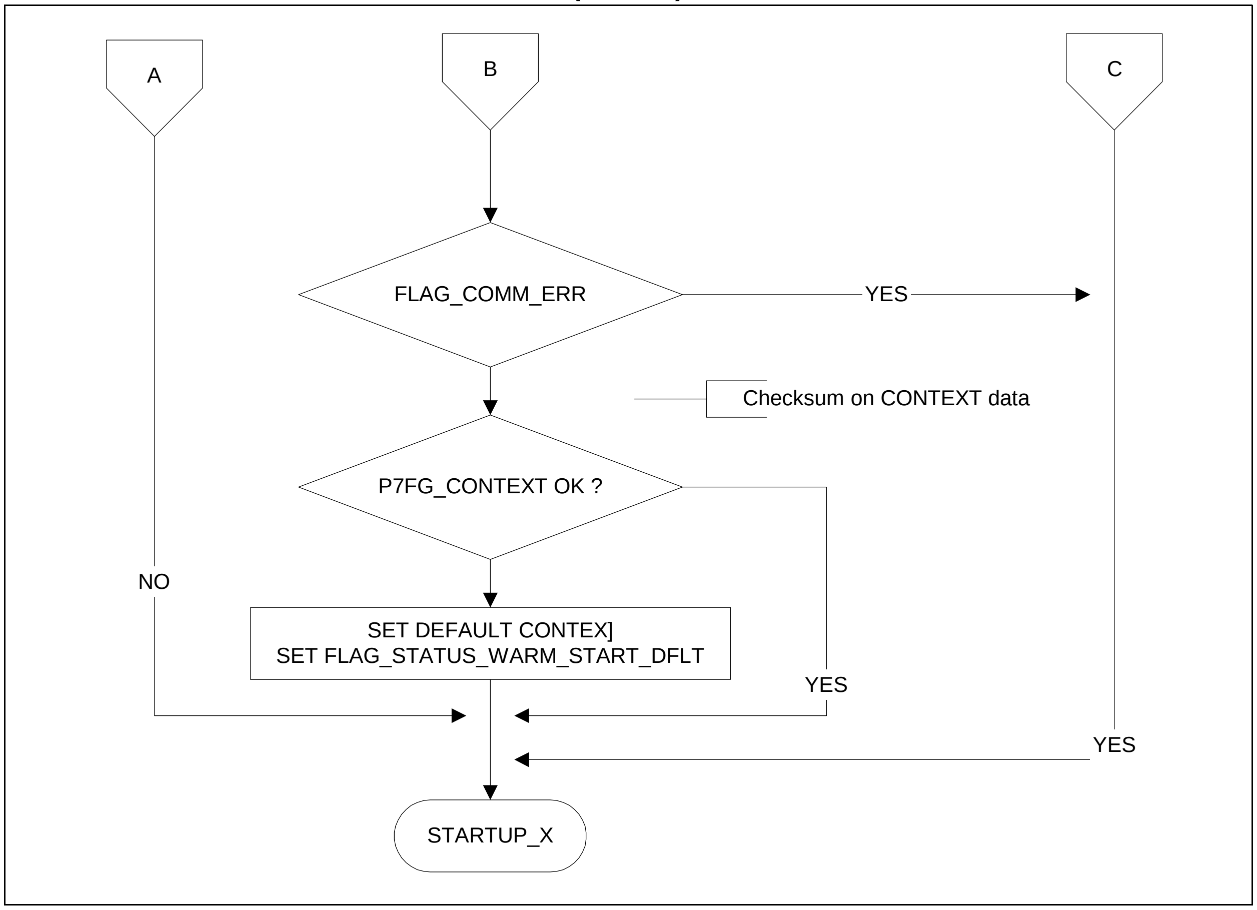

START-UP
FLAG_COMM_ERR
Communication error executing previous command?
RST FLAG_COMM_ERR
SET FLAG_WARM_START
NO
READ P7FG_SERIALNO FROM BATTERY
FLAG_COMM_ERR
User SERIAL NUMBER to identify battery
YES
P7FG_SERIALNO == PREVIOUS
NO
SET FLAG_STATUS_NEW_BATT
FLAG_WARM_START
NO
READ P7FG_CONTEXT FROM BATTERY (P7 RAM)
A
B
C

FIGURE 7-8: START-UP FLOW CHART (PART 2)



7.4.2 BLOCK ADC

During ADC processing, the PS700 ADC Results registers are read individually and processed. Calibration factors and offsets, previously read from PS700 EEPROM during INIT, are applied to the raw data using the following equations. The prefix “CF_” indicates a calibration factor, while the prefix “CO_” indicates a calibration offset. Pack Voltage (VP) is calculated as shown in Equation 7-1.

EQUATION 7-1:

$$VP = (VP\_RAW * CF\_VP)/65536$$

In a two series cell pack, Cell Voltage 2 (VC2) is calculated as shown in Equation 7-2. For a one-cell pack, VC2 and VC2_RAW both equal zero.

EQUATION 7-2:

$$VC2 = (VC2\_RAW * CF\_VC2)/65536$$

Cell Voltage 1 (VC1) is calculated as shown in Equation 7-3.

EQUATION 7-3:

$$VC1 = ((VC1\_RAW + VC2\_RAW) * CF\_VC1)/65536 - VC2$$

Current (I) is calculated as shown in Equation 7-4.

EQUATION 7-4:

$$I = ((I\_RAW - CO\_I) * CF\_I)/65536$$

Temperature (TI) is calculated as shown in Equation 7-5.

EQUATION 7-5:

$$TI = (839 * TI\_RAW)/65536 - 280$$

Figure 7-9 and Figure 7-10 show details of the start-up block.

FIGURE 7-9: ADC FLOW CHART (PART 1)

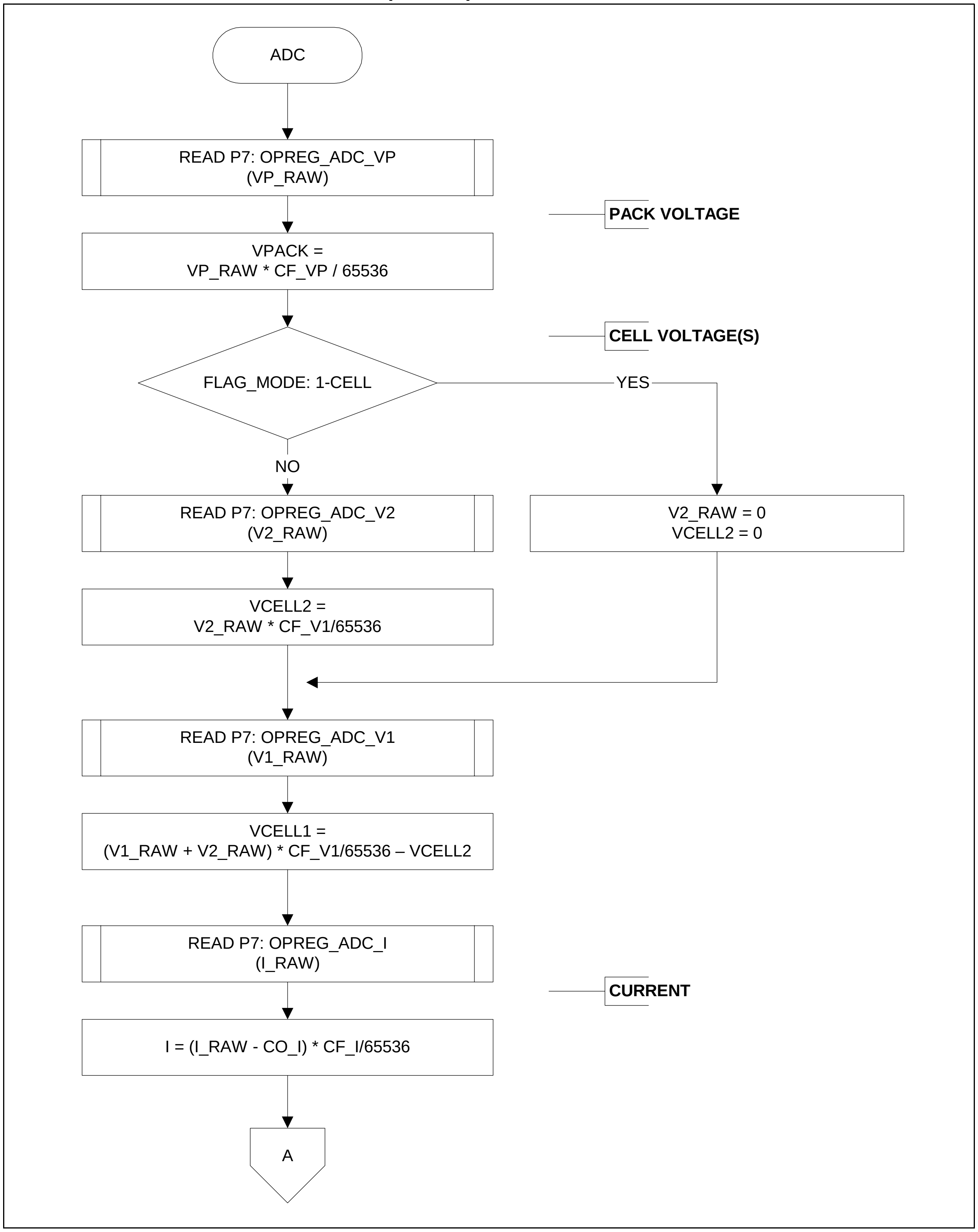
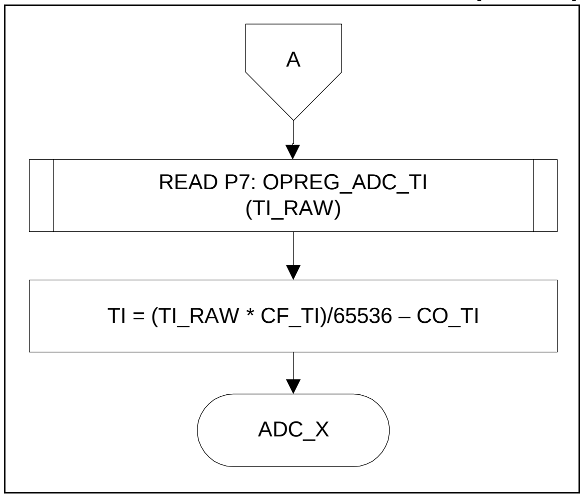

ADC
READ P7: OPREG_ADC_VP (VP_RAW)
PACK VOLTAGE
VPACK = VP_RAW * CF_VP / 65536
CELL VOLTAGE(S)
FLAG_MODE: 1-CELL
YES
NO
READ P7: OPREG_ADC_V2 (V2_RAW)
V2_RAW = 0
VCELL2 = 0
VCELL2 = V2_RAW * CF_V1/65536
READ P7: OPREG_ADC_V1 (V1_RAW)
VCELL1 = (V1_RAW + V2_RAW) * CF_V1/65536 – VCELL2
READ P7: OPREG_ADC_I (I_RAW)
CURRENT
I = (I_RAW - CO_I) * CF_I/65536
A

FIGURE 7-10: ADC FLOW CHART (PART 2)


A
READ P7: OPREG_ADC_TI (TI_RAW)
TI = (TI_RAW * CF_TI)/65536 – CO_TI
ADC_X

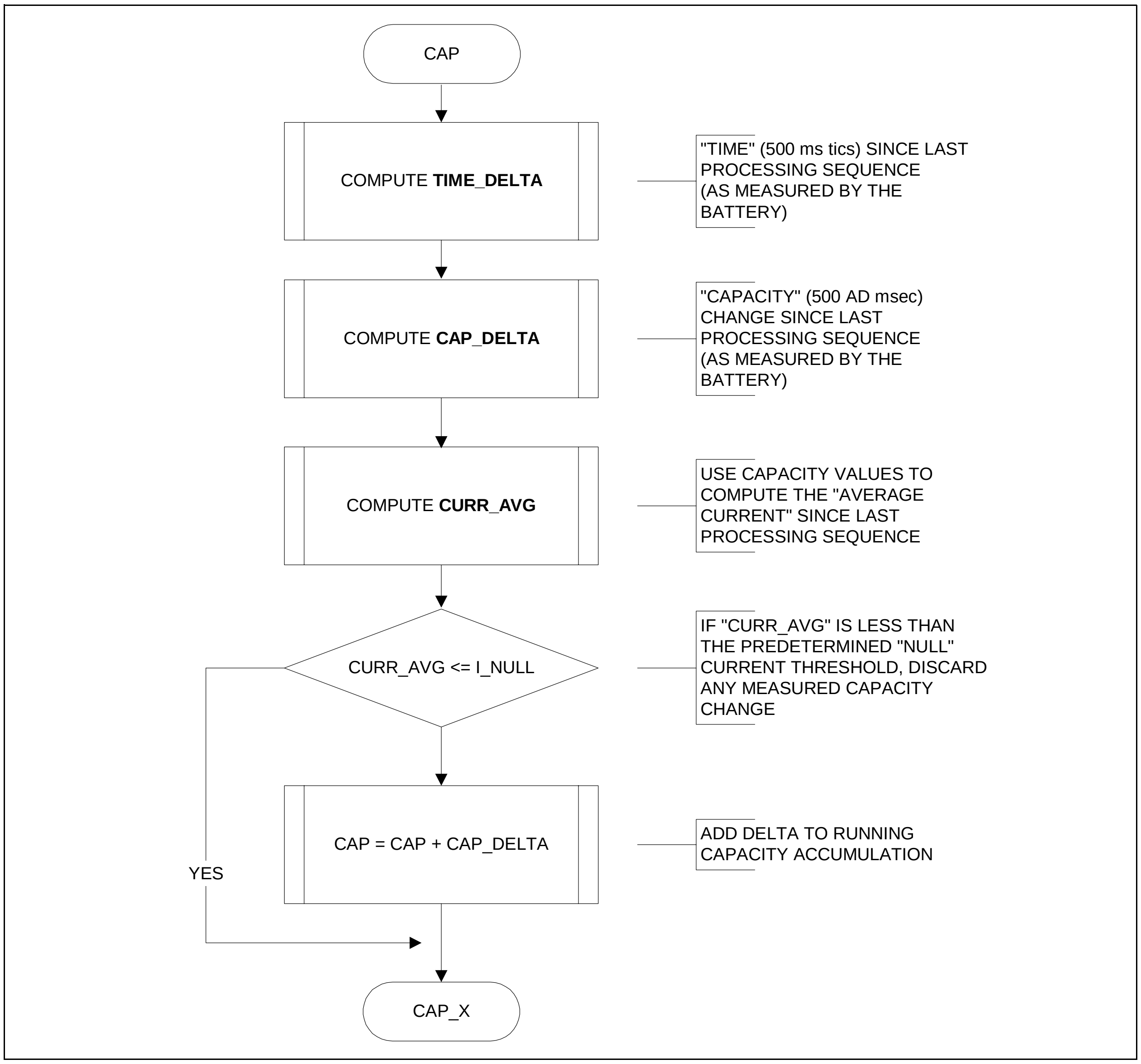
7.4.3 BLOCK CAP

The calculation of remaining capacity is a multi-step process which uses data from the following 32-bit operational registers:

- DTC – Discharge Time Count
- CTC – Charge Time Count
- DCA – Discharge Count Accumulator
- CCA – Charge Count Accumulator

The overall CAP block flow is shown in Figure 7-11.

FIGURE 7-11: CAP FLOW CHART

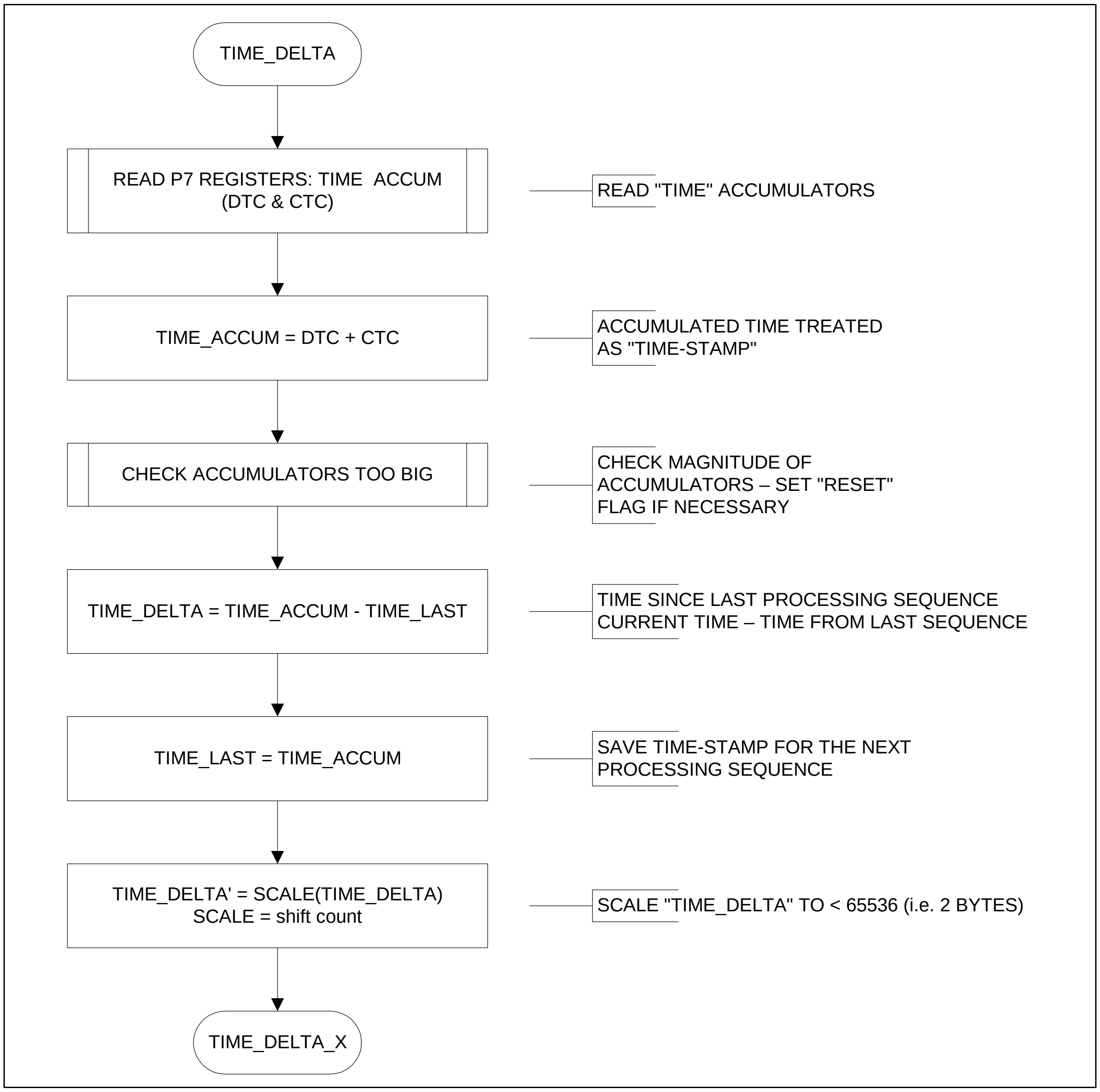

CAP
COMPUTE TIME_DELTA
"TIME" (500 ms tics) SINCE LAST PROCESSING SEQUENCE (AS MEASURED BY THE BATTERY)
COMPUTE CAP_DELTA
"CAPACITY" (500 AD msec) CHANGE SINCE LAST PROCESSING SEQUENCE (AS MEASURED BY THE BATTERY)
COMPUTE CURR_AVG
USE CAPACITY VALUES TO COMPUTE THE "AVERAGE CURRENT" SINCE LAST PROCESSING SEQUENCE
CURR_AVG <= I_NULL
IF "CURR_AVG" IS LESS THAN THE PREDETERMINED "NULL" CURRENT THRESHOLD, DISCARD ANY MEASURED CAPACITY CHANGE
YES
CAP = CAP + CAP_DELTA
ADD DELTA TO RUNNING CAPACITY ACCUMULATION
CAP_X

7.4.3.1 Compute TIME_DELTA

TIME_DELTA is the accumulation of time since the last call to the PS700Driver. The accumulated time (TIME_ACCUM) is computed by summing the values in DTC and CTC. The value of TIME_ACCUM is saved in TIME_LAST at the end of each TIME_DELTA processing sequence. Therefore, the delta time since the last processing instance (TIME_DELTA) is computed by finding the difference between the current accumulation time and the last.

To accommodate the firmware math utilities, operands must fit 16x16 multiply and 32/16 divides. If TIME_DELTA exceeds 16 bits, it is scaled to fit 16 bits with the SCALE used to store the necessary shift count. If TIME_DELTA requires scaling, the time bits lost due to the shifting only represent a 1/65536th of the TIME_DELTA value. This is less than the accuracy of the system and therefore, negligible. See Figure 7-12 for TIME_DELTA computing details.

FIGURE 7-12: TIME_DELTA FLOW CHART

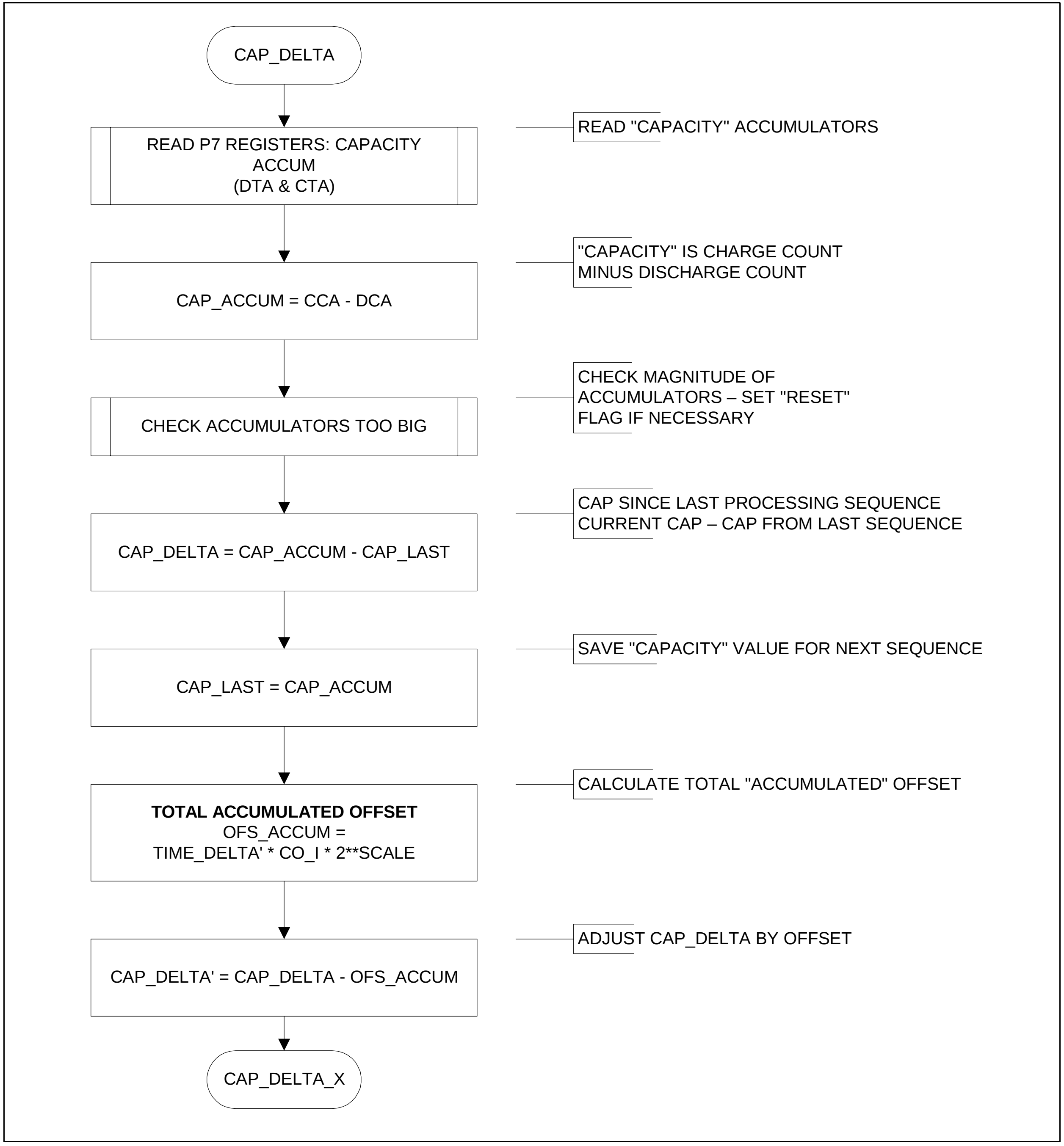

TIME_DELTA
READ P7 REGISTERS: TIME ACCUM (DTC & CTC)
READ "TIME" ACCUMULATORS
TIME_ACCUM = DTC + CTC
ACCUMULATED TIME TREATED AS "TIME-STAMP"
CHECK ACCUMULATORS TOO BIG
CHECK MAGNITUDE OF ACCUMULATORS – SET "RESET" FLAG IF NECESSARY
TIME_DELTA = TIME_ACCUM - TIME_LAST
TIME SINCE LAST PROCESSING SEQUENCE CURRENT TIME – TIME FROM LAST SEQUENCE
TIME_LAST = TIME_ACCUM
SAVE TIME-STAMP FOR THE NEXT PROCESSING SEQUENCE
TIME_DELTA' = SCALE(TIME_DELTA) SCALE = shift count
SCALE "TIME_DELTA" TO < 65536 (i.e. 2 BYTES)
TIME_DELTA_X

7.4.3.2 Compute CAP_DELTA

CAP_DELTA is the difference in capacity as measured by the battery since the last call to the PS700Driver. Accumulated capacity (CAP_ACCUM) is the difference between the charge and discharge accumulators (i.e., CCA – DCA). The value of CAP_ACCUM is saved in CAP_LAST at the end of each CAP_DELTA processing sequence. Therefore, the change in capacity since the last processing instance (CAP_DELTA) is computed by finding the difference between the most recent accumulation of capacity and the last.

The values of the capacity accumulators will include the offset of the most recent measurement. To correct the capacity accumulation measurements for offset, the offset must be multiplied by the delta time (TIME_DELTA). If TIME_DELTA was scaled to fit in 16 bits, the accumulated offset calculation must be corrected to account for the scaling. The CAP_DELTA can then be corrected for the offset by subtracting OFS_ACCUM from CAP_DELTA. CAP_DELTA should then be scaled if necessary. See Figure 7-13 for CAP_DELTA computing details.

FIGURE 7-13: CAP_DELTA FLOW CHART


CAP_DELTA
READ P7 REGISTERS: CAPACITY ACCUM (DTA & CTA)
READ "CAPACITY" ACCUMULATORS
CAP_ACCUM = CCA - DCA
"CAPACITY" IS CHARGE COUNT MINUS DISCHARGE COUNT
CHECK ACCUMULATORS TOO BIG
CHECK MAGNITUDE OF ACCUMULATORS – SET "RESET" FLAG IF NECESSARY
CAP_DELTA = CAP_ACCUM - CAP_LAST
CAP SINCE LAST PROCESSING SEQUENCE CURRENT CAP – CAP FROM LAST SEQUENCE
CAP_LAST = CAP_ACCUM
SAVE "CAPACITY" VALUE FOR NEXT SEQUENCE
TOTAL ACCUMULATED OFFSET OFS_ACCUM = TIME_DELTA' * CO_I * 2**SCALE
CALCULATE TOTAL "ACCUMULATED" OFFSET
CAP_DELTA' = CAP_DELTA - OFS_ACCUM
ADJUST CAP_DELTA BY OFFSET
CAP_DELTA_X

7.4.3.3 Compute CURR_AVG

An average current value is calculated using the calculated accumulated capacity and accumulated time. Since capacity and time information is retrieved from the PS700 asynchronously (serially), it is possible that the communication can straddle the PS700 internal 500 msec processing marker. Since time data is retrieved before capacity data, the time data may be from the previous processing cycle. As a result, the average current may occasionally deviate from the true value. The magnitude of the deviation depends on the processing interval. If the interval is 20 seconds (forty 500 msec tics), the one processing cycle deviation results in a 2.5% jump in average current on these rare occasions. A null current (I_NULL) threshold is applied to the resultant average current. If the computed average current falls below I_NULL, average current is forced to 0 and the prospective delta capacity accumulation is discarded. For ensuing calculations, CAP_DELTA is scaled into 16 bits, SCALE = shift count. The delta capacity (CAP_DELTA) is converted from A/D units to mA(s) by applying the current calibration factor. The scale factor (SCALE) is applied to correct the result.

The delta capacity (CAP_DELTA) is added to the fuel gauge capacity value.

FIGURE 7-14: CURR_AVG FLOW CHART (PART 1)

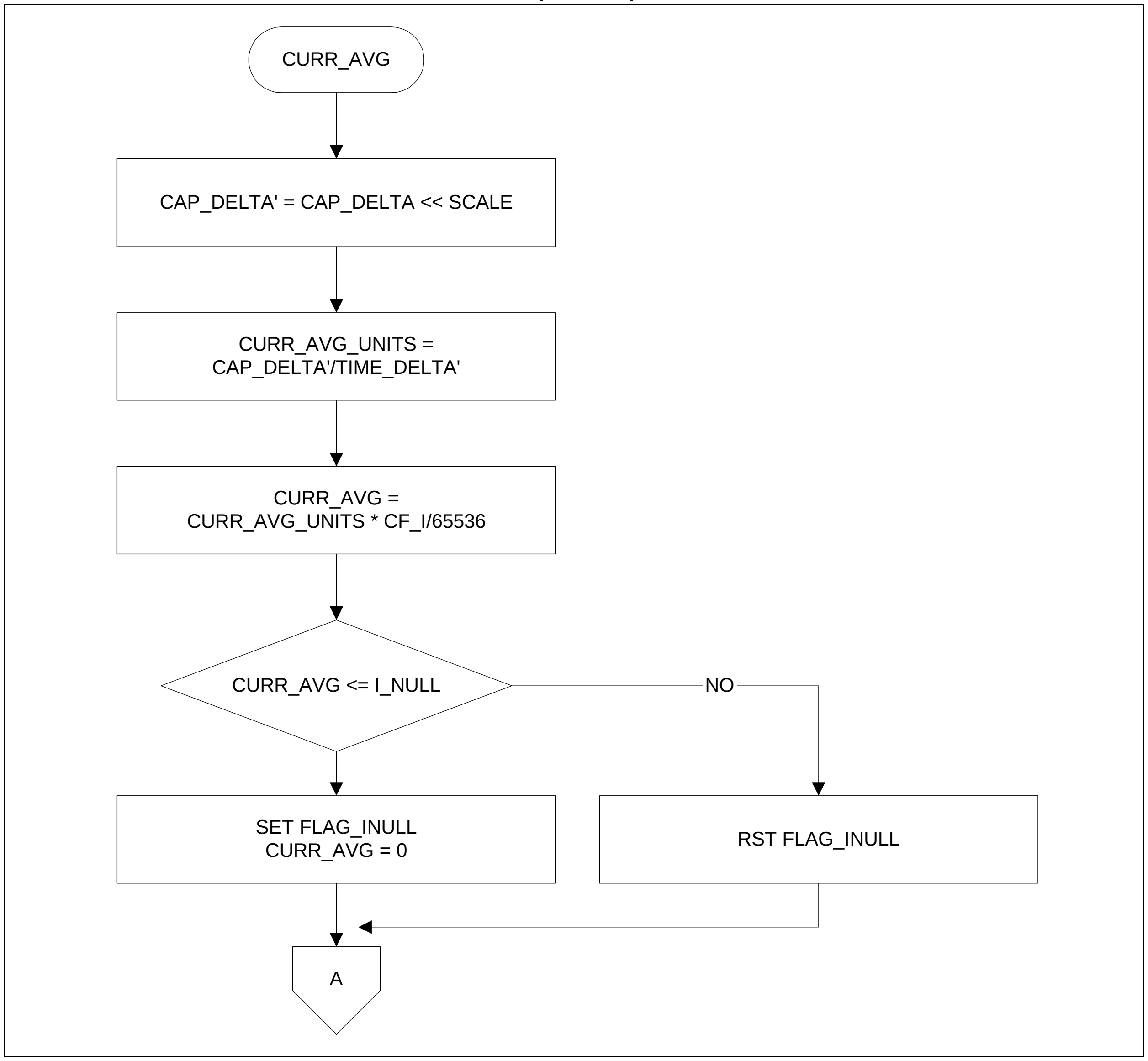
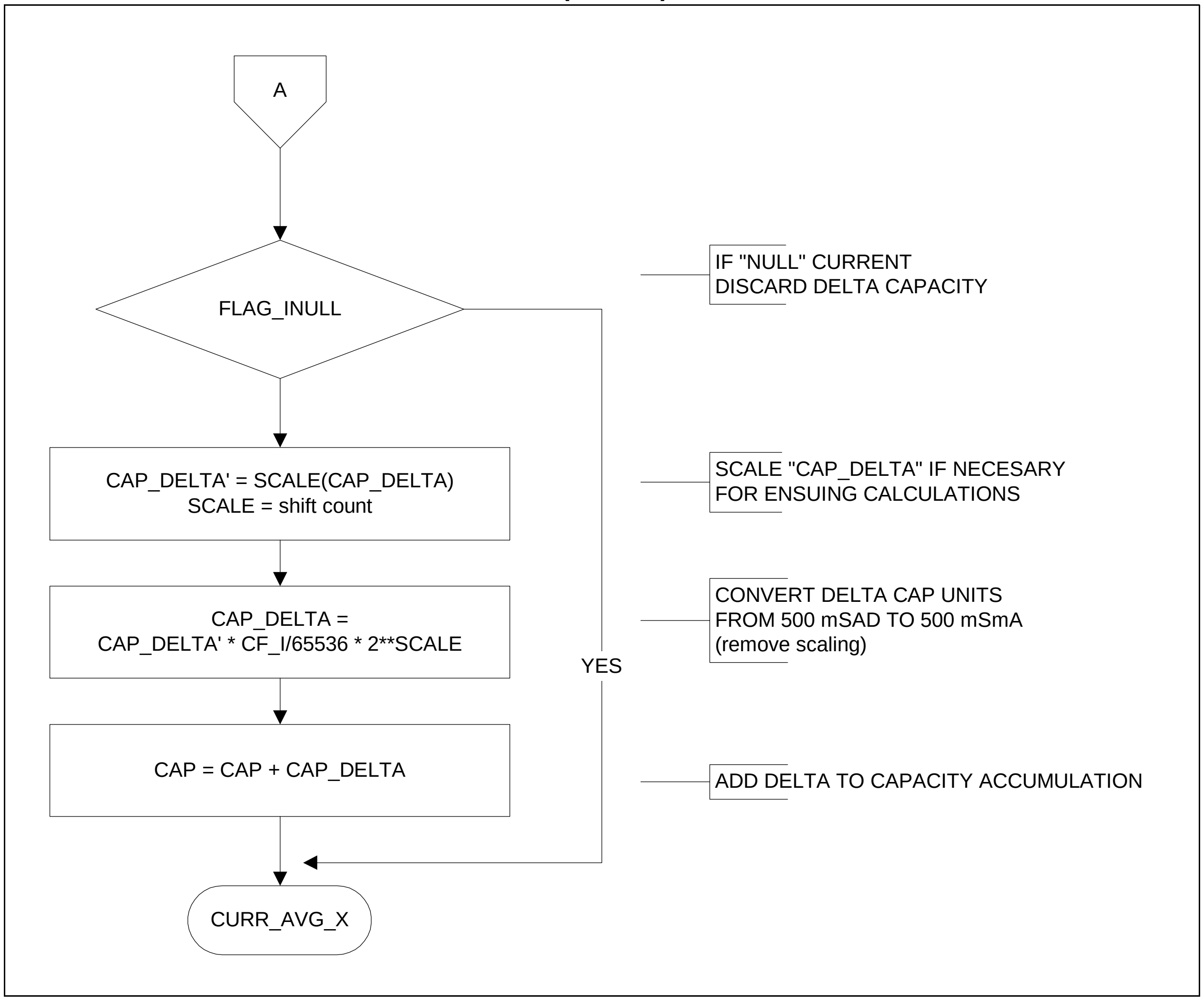

CURR_AVG
CAP_DELTA' = CAP_DELTA << SCALE
CURR_AVG_UNITS = CAP_DELTA'/TIME_DELTA'
CURR_AVG = CURR_AVG_UNITS * CF_I/65536
CURR_AVG <= I_NULL
NO
SET FLAG_INULL CURR_AVG = 0
RST FLAG_INULL
A

FIGURE 7-15: CURR_AVG FLOW CHART (PART 2)



7.4.4 BLOCK SELF-DISCHARGE

Self-discharge is calculated as a function of SOC, temperature and time. At periodic intervals, calculated self-discharge is deducted from the accumulated capacity. The PS700 Temperature Time Count register (TAT) is used to mark time. The Self-Discharge Factor (SDF) is a function of SOC and temperature. The SDF can be derived from a Look-up Table (LUT) or calculated by Equation 7-6. The SDF units are 1/256 of Full Charge Capacity (CAP_FULL).

EQUATION 7-6:

$$SDF = A + ((T - 15) + (S - 70))/10 * B \text{ [Limit: 255]}$$

where:

A = constant (= 2)

B = constant (= 2)

T = temperature [degC + 20]

S = State-Of-Charge [%]

Using the equation to calculate SDF results in the values in Table 7-3.

TABLE 7-3: SDF VALUES FROM EQUATION 7-6

		Temperature (degC)							
		15	**25**	**32**	**38**	**44**	**50**	**55**	**>55**
SOC (%)	**73**	4	6	6	10	10	18	34	34
	78	4	6	6	10	10	18	34	34
	83	6	10	10	18	18	34	66	66
	88	6	10	10	18	18	34	66	66
	92	10	18	18	34	34	66	130	130
	95	10	18	18	34	34	66	130	130
	98	10	18	18	34	34	66	130	130
	>98	18	34	34	66	66	130	255	255

FIGURE 7-16: SELF-DISCHARGE FLOW CHART (PART 1)

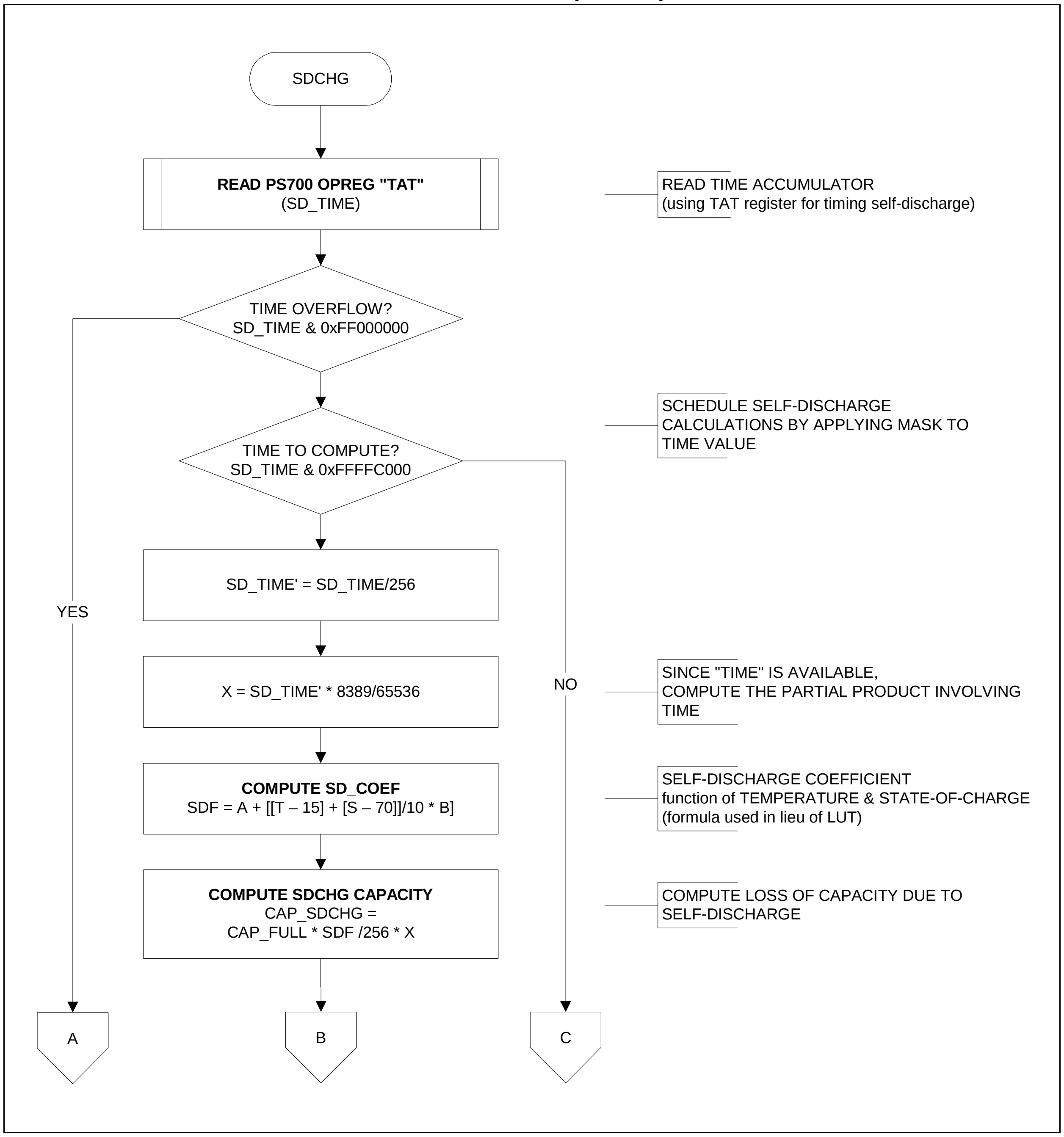
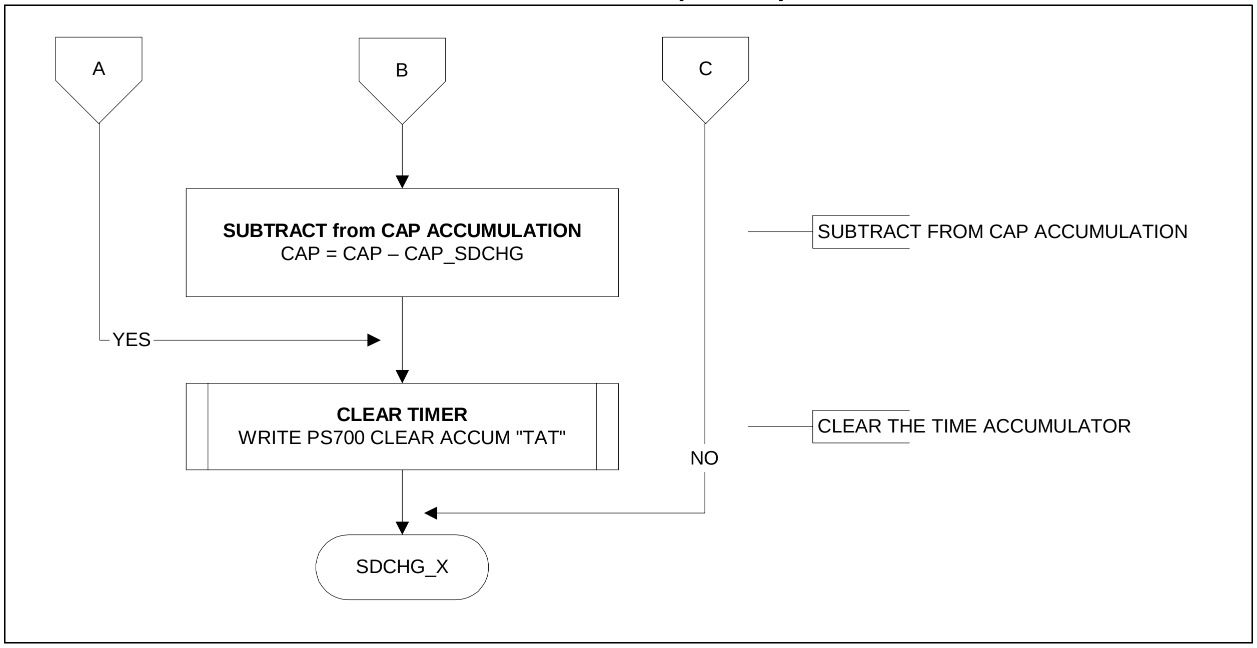

SDCHG
READ PS700 OPREG "TAT"
(SD_TIME)
READ TIME ACCUMULATOR
(using TAT register for timing self-discharge)
TIME OVERFLOW?
SD_TIME & 0xFF000000
YES
TIME TO COMPUTE?
SD_TIME & 0xFFFFC000
NO
SCHEDULE SELF-DISCHARGE CALCULATIONS BY APPLYING MASK TO TIME VALUE
SD_TIME' = SD_TIME/256
X = SD_TIME' * 8389/65536
SINCE "TIME" IS AVAILABLE, COMPUTE THE PARTIAL PRODUCT INVOLVING TIME
COMPUTE SD_COEF
SDF = A + [[T – 15] + [S – 70]]/10 * B]
SELF-DISCHARGE COEFFICIENT function of TEMPERATURE & STATE-OF-CHARGE (formula used in lieu of LUT)
COMPUTE SDCHG CAPACITY
CAP_SDCHG = CAP_FULL * SDF /256 * X
COMPUTE LOSS OF CAPACITY DUE TO SELF-DISCHARGE
A
B
C

FIGURE 7-17: SELF-DISCHARGE FLOW CHART (PART 2)


A
B
C
SUBTRACT from CAP ACCUMULATION
CAP = CAP – CAP_SDCHG
SUBTRACT FROM CAP ACCUMULATION
YES
CLEAR TIMER
WRITE PS700 CLEAR ACCUM "TAT"
CLEAR THE TIME ACCUMULATOR
NO
SDCHG_X

Knowing the SDF, the capacity lost due to self-discharge in internal units (500 mSmA) is calculated using Equation 7-7.

EQUATION 7-7:

CAP_SD_UNITS = P/1000 * SDF/256 * TIME/172800 * FCC * 7200

where:

P = 10ths of percent of FCC corresponding to SDF = 256

TIME = elapsed time in [500 mS]

FCC = full charge capacity [mAh]

Typically, SDF is scaled so that SDF = 256 correlates to 1.2% FCC per day (i.e., P = 12 above) resulting in Equation 7-8.

EQUATION 7-8:

CAP_SD_UNITS = 12/1000 * SDF/256 * TIME/172800 * FCC * 7200

Because self-discharge is a small value, this calculation can be scheduled at long time intervals. TIME can be modified to TIME', where TIME' = TIME/256 (i.e., units = 128 seconds). This allows a 16-bit elapsed time accumulator to represent 128 seconds to 97 days. Anything in excess of 97 days is declared to be an overflow, so the self-discharge calculation is ignored and the elapsed time accumulator is reset. Each time self-discharge is calculated, the elapsed time accumulator is reset. This results in the loss of some fraction of 128 seconds in accumulated time. For this loss to be a small percentage of the accumulated time, SDCHG is scheduled every 64 counts of TIME' (i.e., 64 * 128 sec = 2.3 hours). The firmware implementation of self-discharge, using integer math and minimizing division, results in Equation 7-9.

EQUATION 7-9:

CAP_SD_UNITS = ((8289 * TIME')/65536) * ((FCC * SDF)/256)

7.4.5 BLOCK CAP_CALC

At this stage capacity (internal units) has been determined based on the battery capacity accumulators and self-discharge calculations. The following derivative capacity-based data can now be calculated:

1. Residual capacity – unusable capacity due to temperature and discharge rate.
2. Reported capacity – capacity converted from 500 mSmA to mAh(s).
3. Relative SOC – capacity as a percentage of FCC.
4. Cycle count.

7.4.5.1 Residual Capacity

Residual capacity (RCAP) is defined as that portion of battery capacity that is unavailable to the system based on discharge rate and temperature. RCAP has been implemented at a constant discharge rate. This discharge rate is the rate that the system will discharge the battery when performing shutdown procedures. The PS700Driver uses a Look-up table (see Table 7-4) to define RCAP in eight temperature zones. As seen in the EEPROM memory map (Table 6-1), RCAP_T defines the temperature zones and RCAP defines the residual capacity per zone.

TABLE 7-4: RCAP LUT

<RCAP_T[0]	<RCAP_T[1]	<RCAP_T[2]	<RCAP_T[3]	<RCAP_T[4]	<RCAP_T[5]	<RCAP_T[6]	>RCAP_T[6]
RCAP[0]	RCAP[1]	RCAP[2]	RCAP[3]	RCAP[4]	RCAP[5]	RCAP[6]	RCAP[7]

The values of RCAP_T are calculated using Equation 7-10.

EQUATION 7-10:

$$RCAP\_T = TEMPERATURE + 20°C$$

For example, a RCAP_T value of 0 corresponds to a temperature of -20 degrees Celsius. The residual capacity is calculated using Equation 7-11.

EQUATION 7-11:

$$CAP\_RES = RCAP * CAP\_FULL/256$$

It is calculated only when temperature changes by DELTA_TEMP. When capacity compensation is enabled (see Table 6-5, Mode Map), the calculated residual capacity (CAP_RES) is deducted from internal capacity (CAP_UNITS) to yield reported capacity (CAP).

7.4.6 BLOCK CHG/DCHG

The capacity-based average current value is used to indicate whether the battery is charging, known as the Charge Increasing (CI) state, or discharging, known as the Charge Decreasing (CD) state.

7.4.6.1 CI

During CI, data is examined for the lithium ion/polymer End-Of-Charge (EOC) condition. One or more cells must have a voltage greater than the value of VEOC and the current must be greater than zero and less than the IEOC parameter. EOC is not valid until the condition exists for 2 consecutive calls to PS700.

FIGURE 7-18: CI FLOW CHART (PART 1)

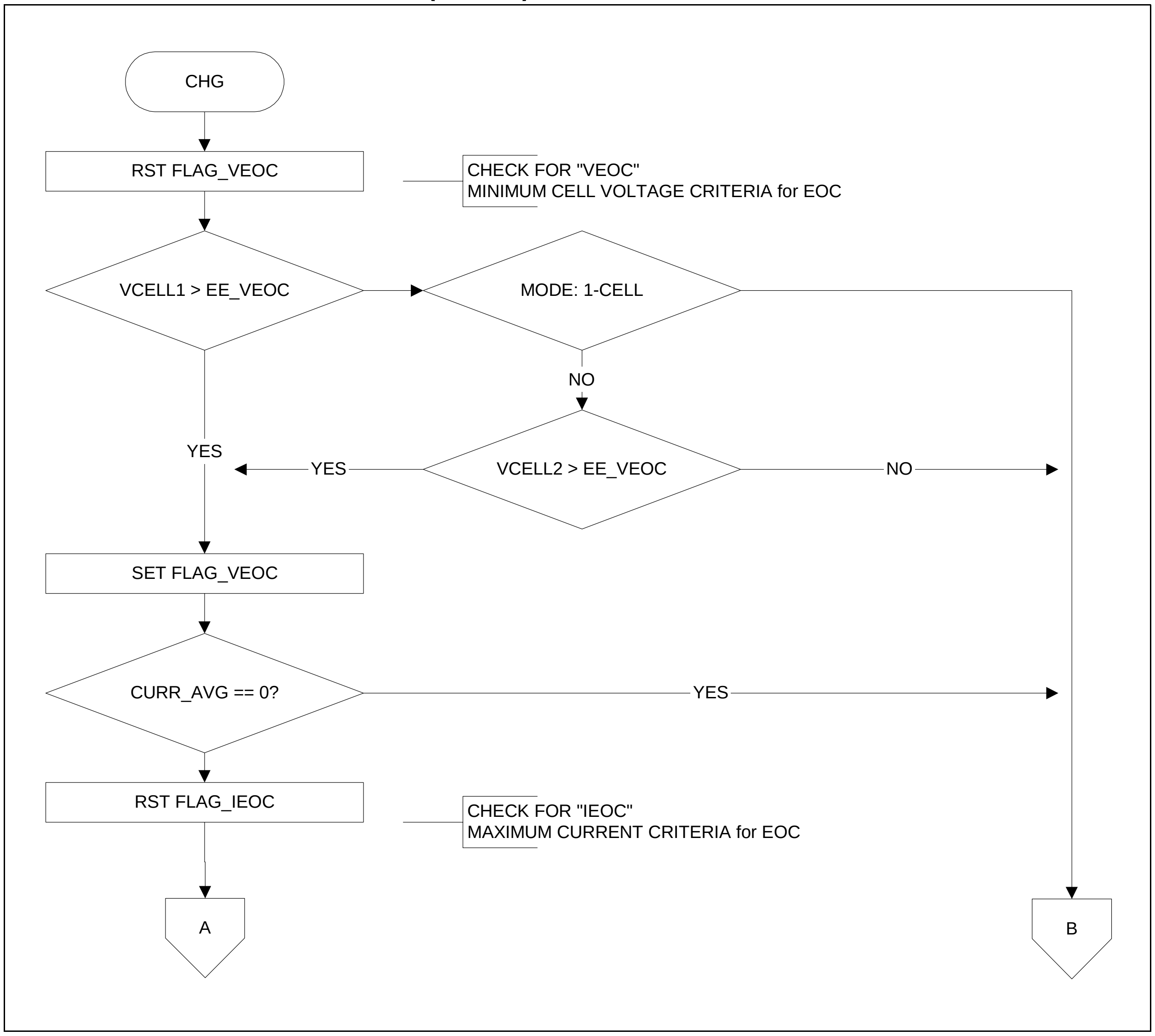
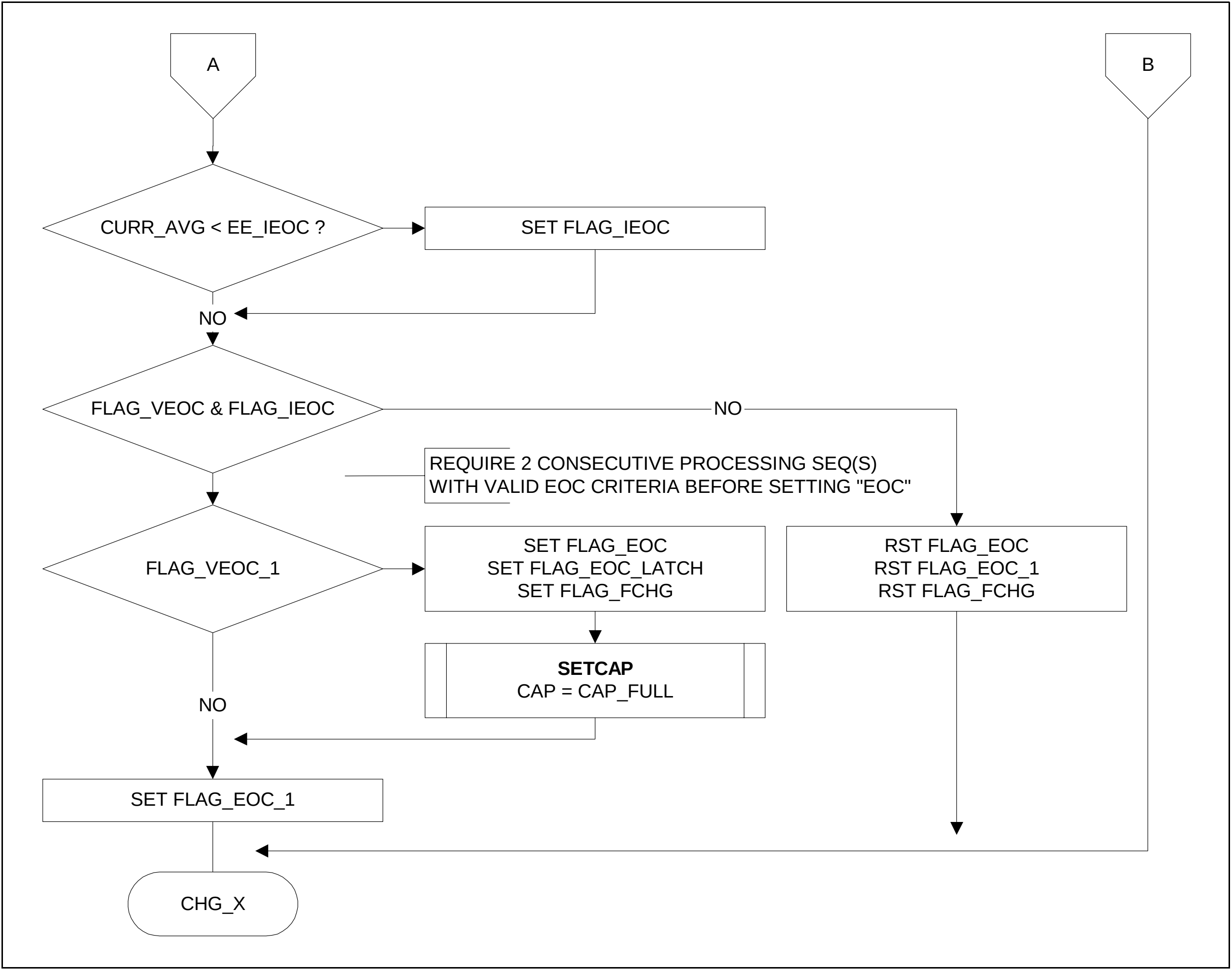


FIGURE 7-19: CI FLOW CHART (PART 2)



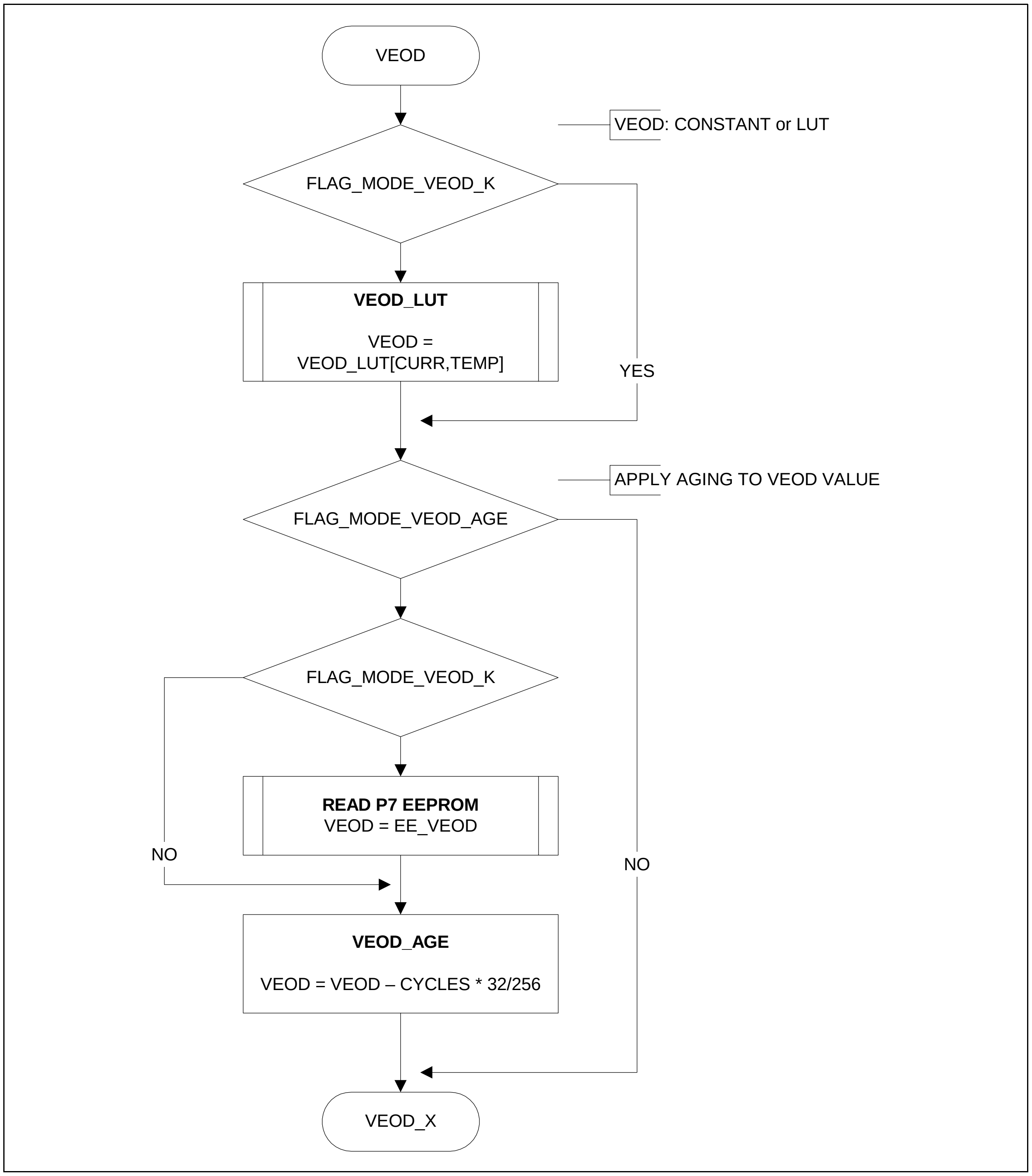
7.4.6.2 CD

During CD, data is examined for the End-Of-Discharge (EOD) condition and capacity information is processed. One or more cells must have a voltage less than the value of VEOD for 2 consecutive calls to PS700. The value of VEOD can be a constant over all rates and temperatures, or dynamically derived from a look-up table (Table 7-5).

TABLE 7-5: VEOD_LUT

	<C[0]	**<C[1]**	**<C[2]**	**>C[2]**
<T[0]	VEOD[0,0]	VEOD[1,0]	VEOD[2,0]	VEOD[3,0]
<T[1]	VEOD[0,1]	VEOD[1,1]	VEOD[2,1]	VEOD[3,1]
<T[2]	VEOD[0,2]	VEOD[1,2]	VEOD[2,2]	VEOD[3,2]
<T[3]	VEOD[0,3]	VEOD[1,3]	VEOD[2,3]	VEOD[3,3]
<T[4]	VEOD[0,4]	VEOD[1,4]	VEOD[2,4]	VEOD[3,4]
<T[5]	VEOD[0,5]	VEOD[1,5]	VEOD[2,5]	VEOD[3,5]
<T[6]	VEOD[0,6]	VEOD[1,6]	VEOD[2,6]	VEOD[3,6]
>T[6]	VEOD[0,7]	VEOD[1,7]	VEOD[2,7]	VEOD[3,7]

FIGURE 7-20: VEOD FLOW CHART



The VEOD LUT is a 2-D matrix of voltages organized in 4 discharge rate zones (columns) and 8 temperature zones (rows). The VEOD_LUT values are calculated using the following equations:

EQUATION 7-12:

$$C = CURR\_AVG/16$$

EQUATION 7-13:

$$T = TEMPERATURE + 20$$

EQUATION 7-14:

$$VEOD = VEOD[C,T] * 4 + 2600$$

FIGURE 7-21: VEOD_LUT FLOW CHART (PART 1)

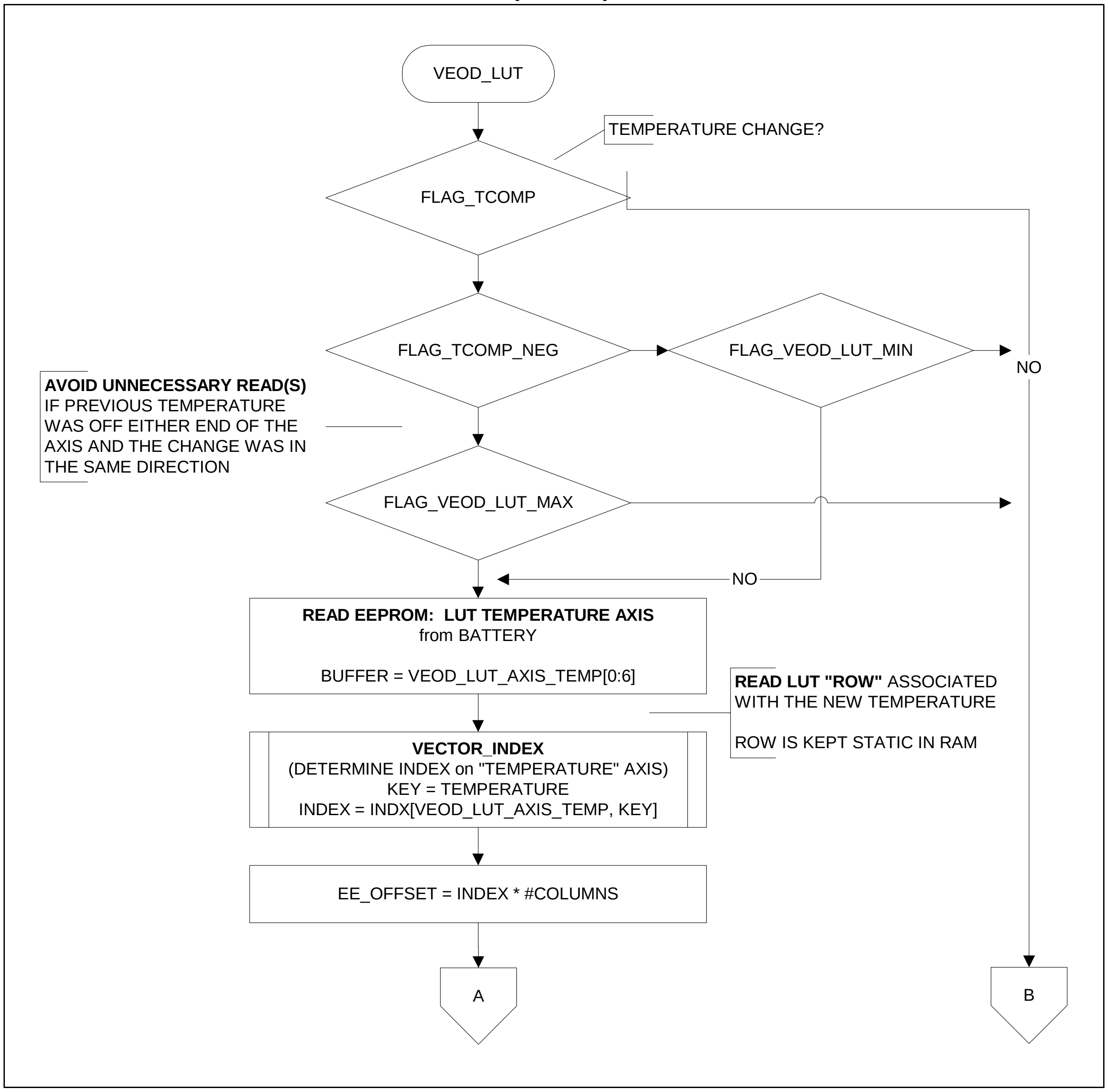
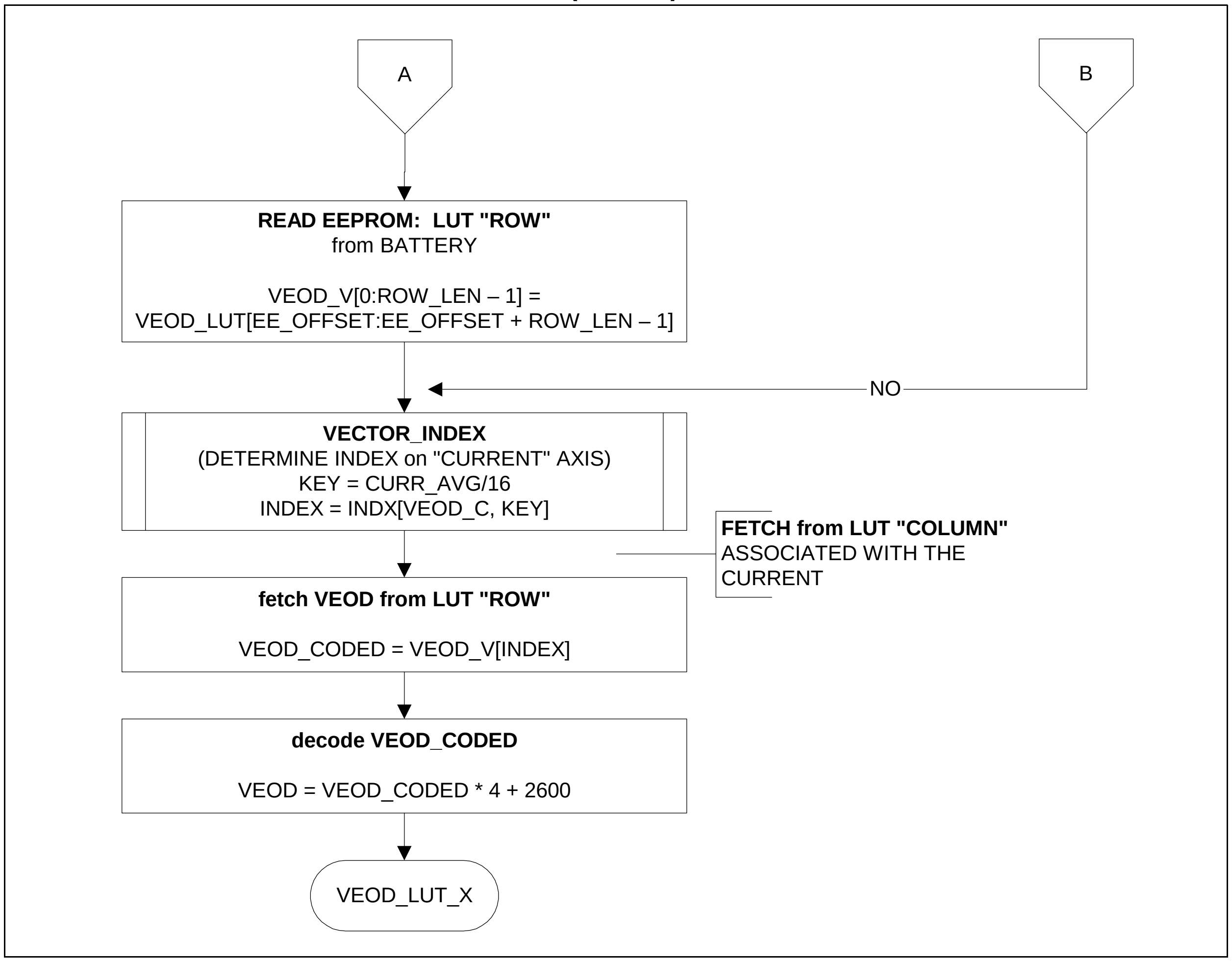

VEOD_LUT
TEMPERATURE CHANGE?
FLAG_TCOMP
FLAG_TCOMP_NEG
FLAG_VEOD_LUT_MIN
NO
AVOID UNNECESSARY READ(S)
IF PREVIOUS TEMPERATURE WAS OFF EITHER END OF THE AXIS AND THE CHANGE WAS IN THE SAME DIRECTION
FLAG_VEOD_LUT_MAX
NO
READ EEPROM: LUT TEMPERATURE AXIS from BATTERY
BUFFER = VEOD_LUT_AXIS_TEMP[0:6]
READ LUT "ROW" ASSOCIATED WITH THE NEW TEMPERATURE
ROW IS KEPT STATIC IN RAM
VECTOR_INDEX
(DETERMINE INDEX on "TEMPERATURE" AXIS)
KEY = TEMPERATURE
INDEX = INDX[VEOD_LUT_AXIS_TEMP, KEY]
EE_OFFSET = INDEX * #COLUMNS
A
B

FIGURE 7-22: VEOD_LUT FLOW CHART (PART 2)


A
B
READ EEPROM: LUT "ROW"
from BATTERY
VEOD_V[0:ROW_LEN – 1] =
VEOD_LUT[EE_OFFSET:EE_OFFSET + ROW_LEN – 1]
NO
VECTOR_INDEX
(DETERMINE INDEX on "CURRENT" AXIS)
KEY = CURR_AVG/16
INDEX = INDX[VEOD_C, KEY]
FETCH from LUT "COLUMN"
ASSOCIATED WITH THE
CURRENT
fetch VEOD from LUT "ROW"
VEOD_CODED = VEOD_V[INDEX]
decode VEOD_CODED
VEOD = VEOD_CODED * 4 + 2600
VEOD_LUT_X

The EOD threshold (VEOD) can be optionally modified by an aging factor based on cycle count.

EQUATION 7-15:

$$VEOD' = VEOD - CYCLES * AGE\_FACTOR/256$$

where:

VEOD' = new VEOD

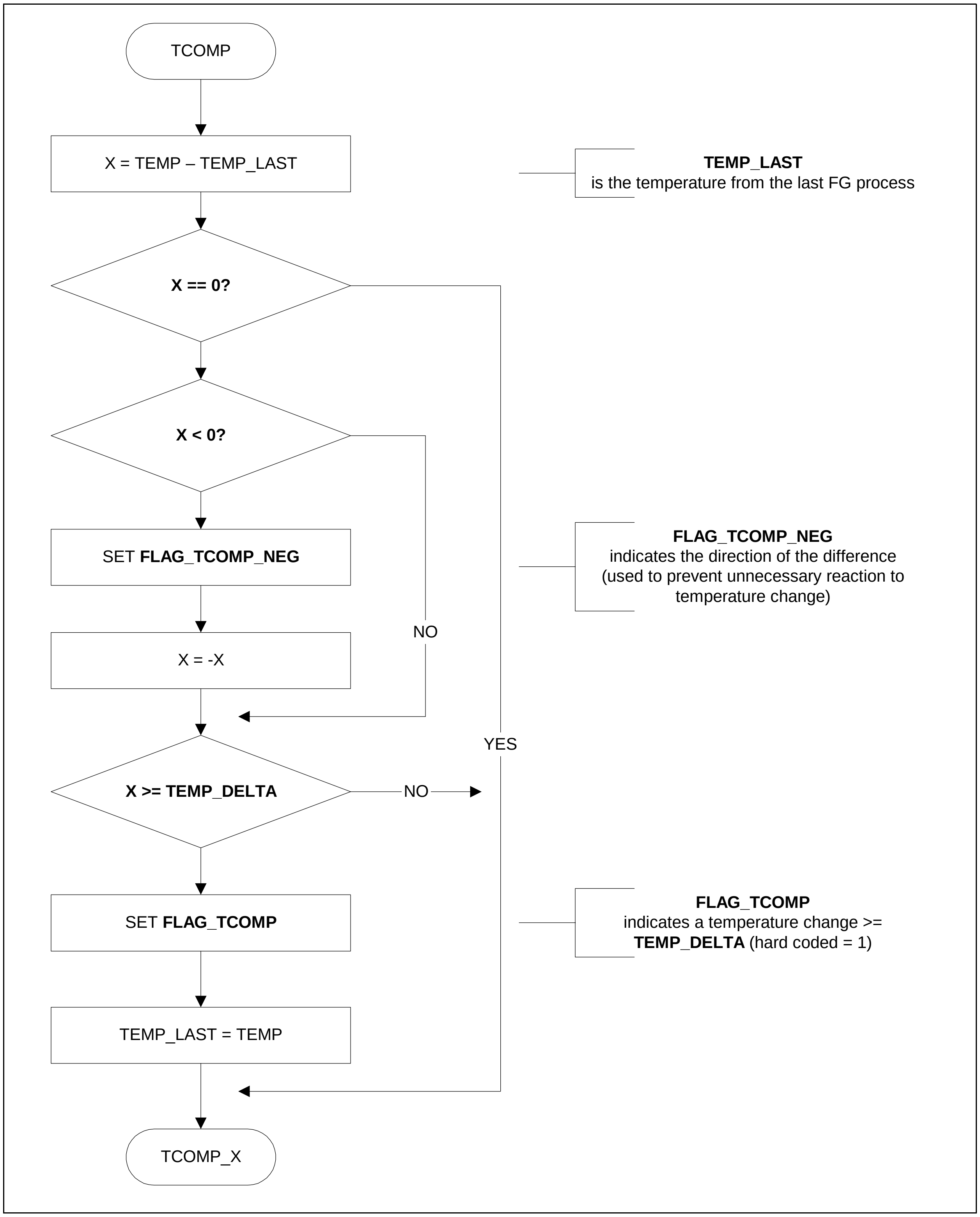
VEOD = old VEOD

CYCLES = cycle count

AGE_FACTOR = predetermined effect of cycles on EOD voltage (hard coded = 32)

To minimize unnecessary or redundant communication with the battery, a VEOD_LUT row is kept in system RAM and only retrieved when the temperature changes by a predetermined value DELTA_TEMP. While temperature remains constant, VEOD can be determined from existing data in system RAM based on discharge rate. A flag is also set to indicate the direction of temperature change, more negative or more positive. This further reduces communication with the PS700 when the new temperature is in the same zone. When the new temperature is in a new zone, the applicable row is retrieved from the battery and stored in RAM. During each processing loop, the VEOD can be determined, using the discharge rate from the LUT row in system RAM. The details of the data retrieval are shown in Figure 7-23.

FIGURE 7-23: TEMPERATURE CHANGE FLOW CHART


TCOMP
X = TEMP – TEMP_LAST
X == 0?
X < 0?
SET FLAG_TCOMP_NEG
X = -X
NO
YES
X >= TEMP_DELTA
NO
SET FLAG_TCOMP
TEMP_LAST = TEMP
TCOMP_X
TEMP_LAST is the temperature from the last FG process
FLAG_TCOMP_NEG indicates the direction of the difference (used to prevent unnecessary reaction to temperature change)
FLAG_TCOMP indicates a temperature change >= TEMP_DELTA (hard coded = 1)

At EOD, the PPS700Driver can optionally learn a new full charge capacity (FCC or CAP_FULL). The fuel gauge will learn if the battery maintained a CD state since detecting EOC. Any intervening charge since EOC will prevent FCC learning. The measured capacity between EOC and EOD is used to determine a new CAP_FULL, as shown in Equation 7-16.

EQUATION 7-16:

$$CAP\_FULL' = CAP\_FULL - CAP + CAP\_RES + CAP\_EOD$$

where:

CAP_FULL'	= new full charge capacity
CAP_FULL	= old full charge capacity
CAP	= capacity at EOD
CAP_RES	= residual capacity
CAP_EOD	= arbitrary capacity around which the VEOD table was built

FIGURE 7-24: EOD_CAP FLOW CHART (PART 1)

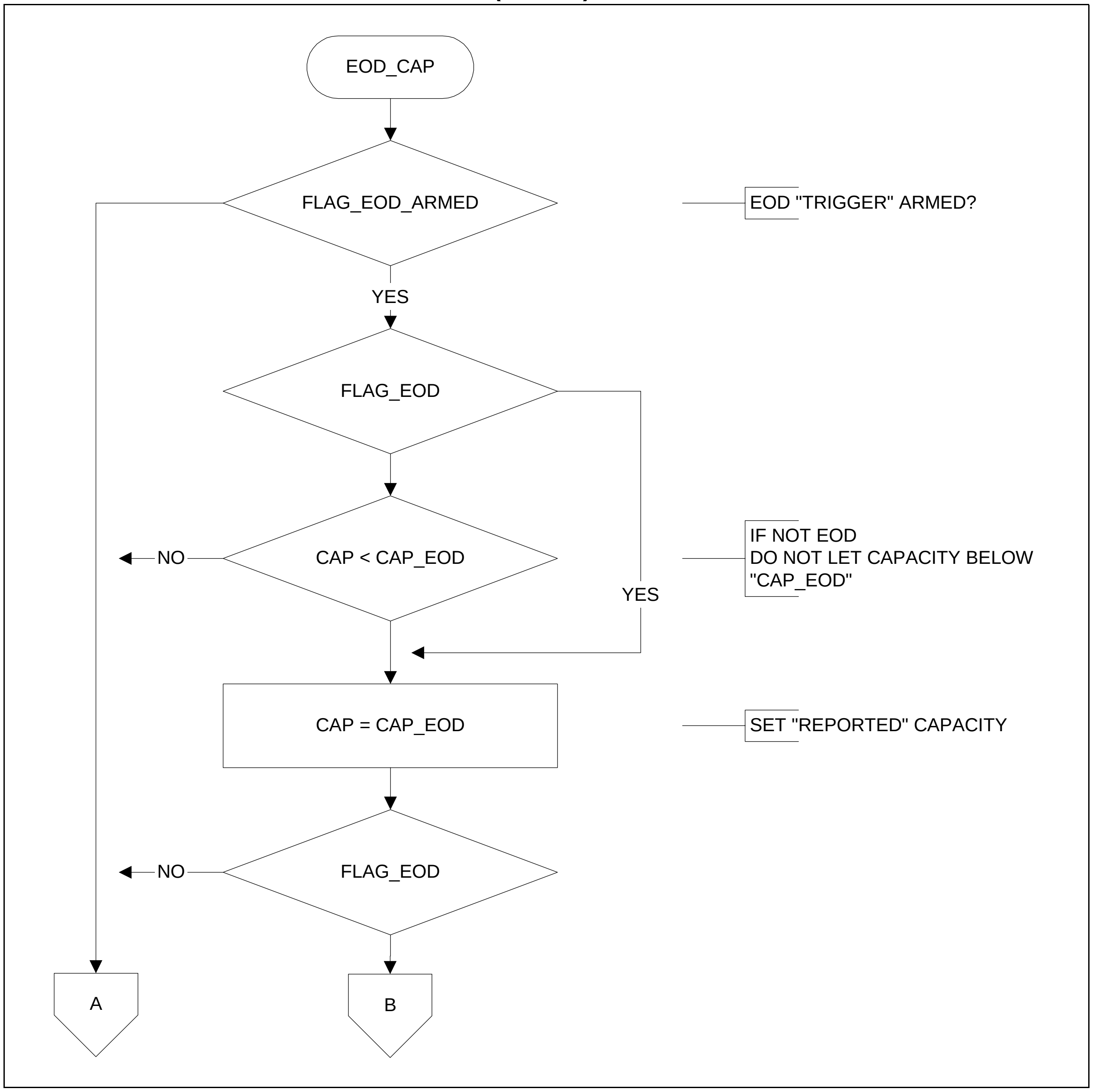


FIGURE 7-25: EOD_CAP FLOW CHART (PART 2)

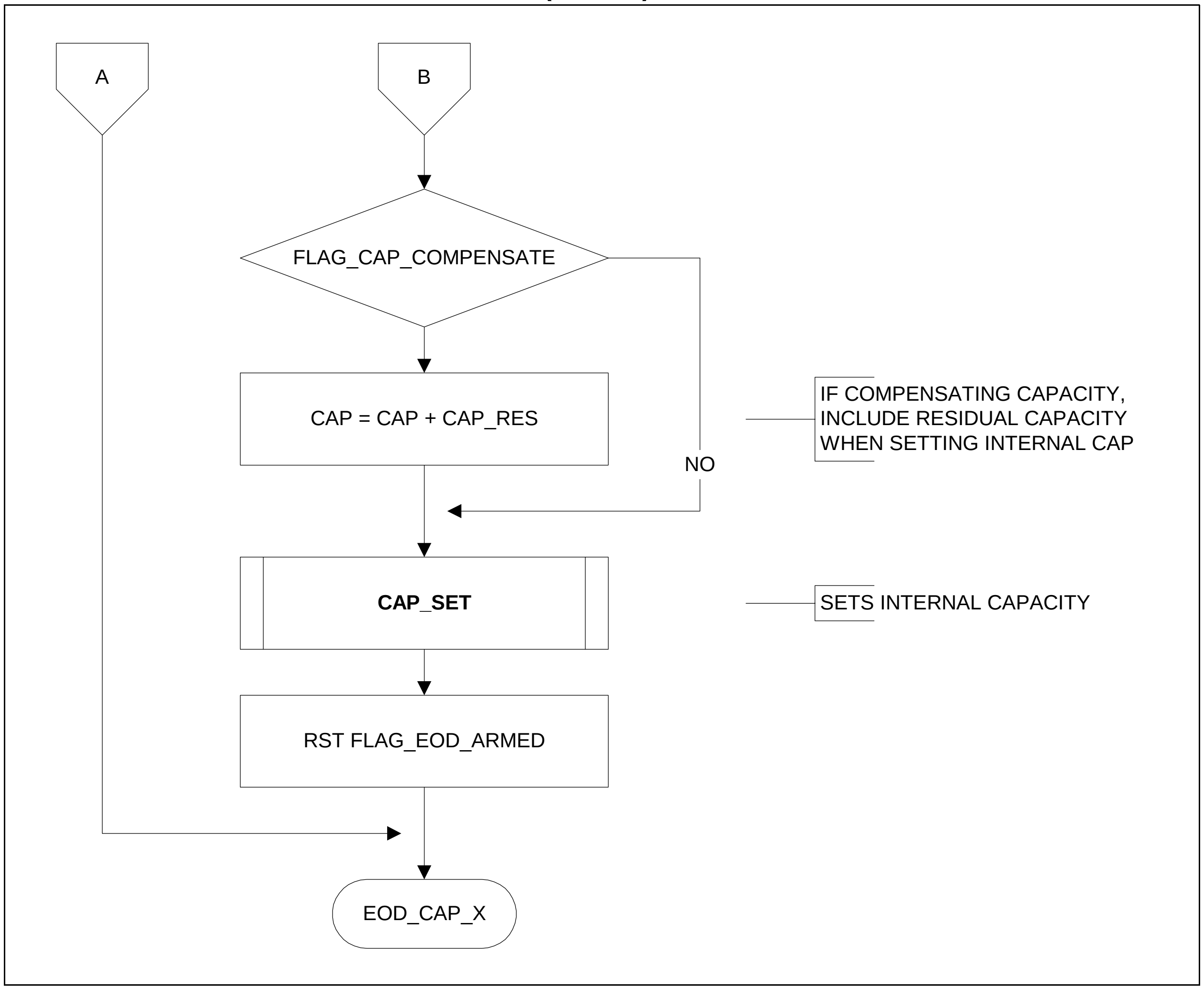


FIGURE 7-26: CD FLOW CHART (PART 1)

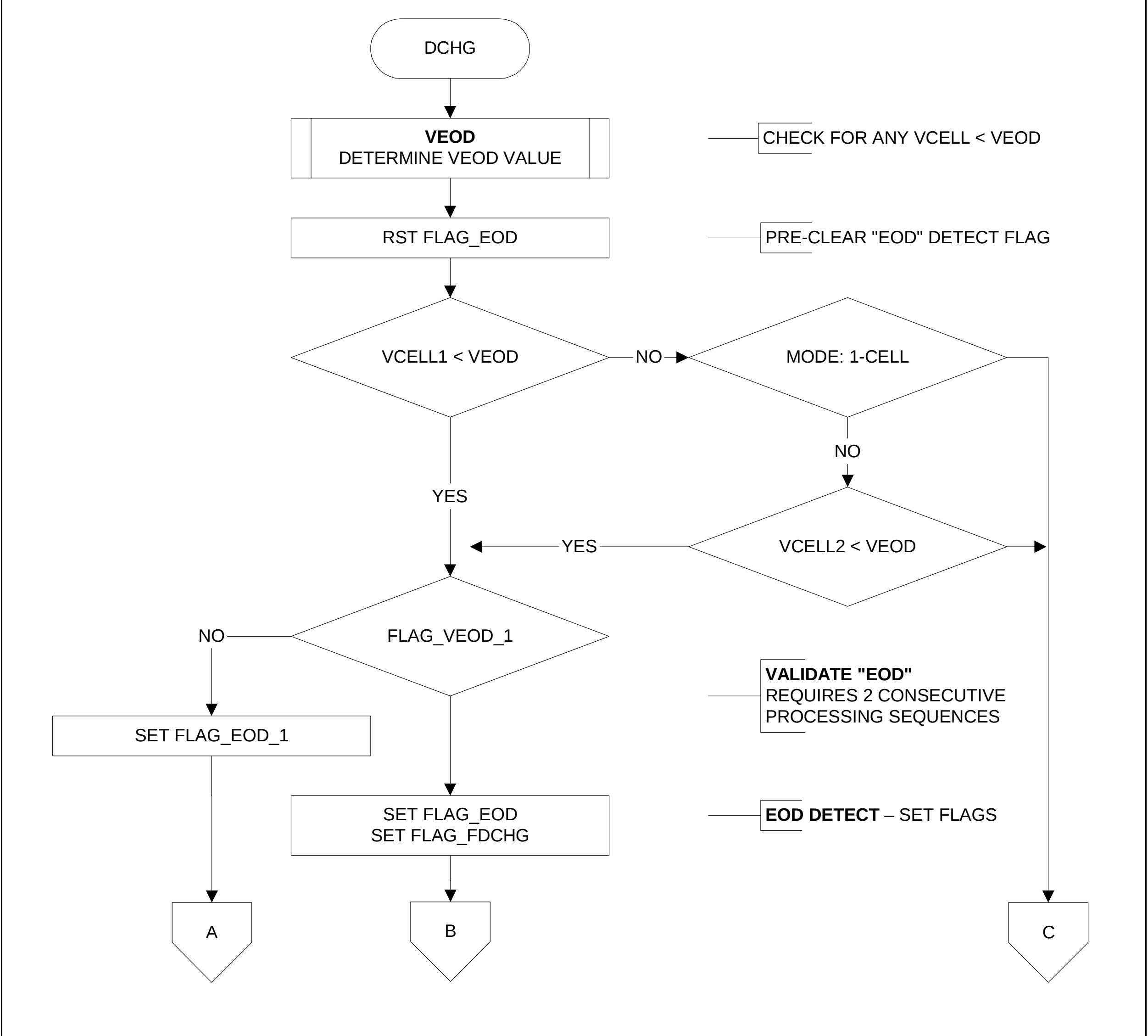
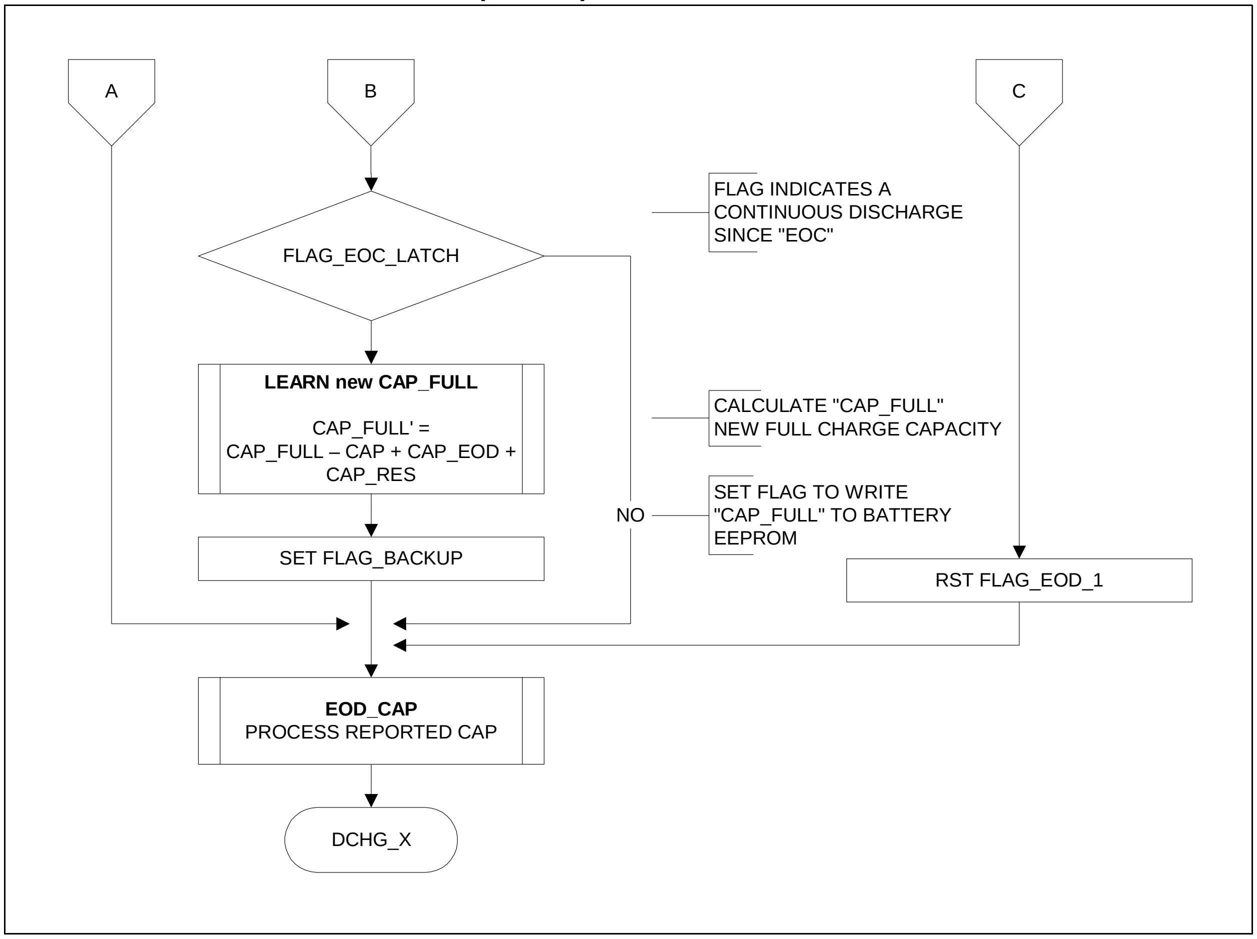


FIGURE 7-27: CD FLOW CHART (PART 2)

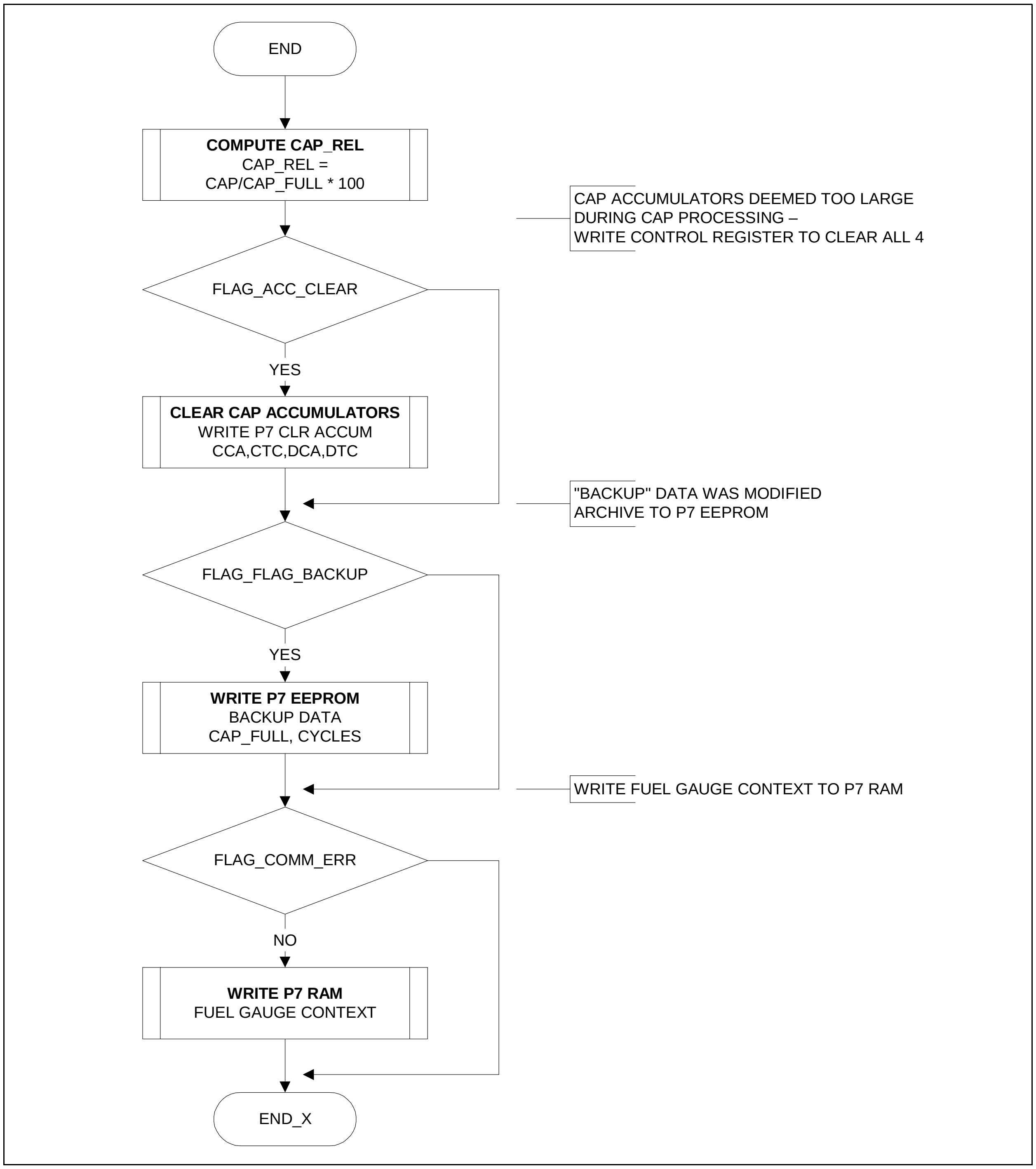

A
B
C
FLAG_EOC_LATCH
FLAG INDICATES A CONTINUOUS DISCHARGE SINCE "EOC"
LEARN new CAP_FULL
CAP_FULL' = CAP_FULL – CAP + CAP_EOD + CAP_RES
CALCULATE "CAP_FULL" NEW FULL CHARGE CAPACITY
NO
SET FLAG TO WRITE "CAP_FULL" TO BATTERY EEPROM
SET FLAG_BACKUP
RST FLAG_EOD_1
EOD_CAP
PROCESS REPORTED CAP
DCHG_X

7.4.7 BLOCK END

The post-process section performs functions to finalize processing and prepare for the next call.

1. Clear Accumulators – if capacity accumulators (time or capacity) were deemed too large (at risk of overflow), they are reset by writing to the Accumulator Clear register.
2. Data Backup – if data to be archived was modified during processing (e.g., CYCLE_COUNT, CAP_FULL), the new/modified value is written to PS700 EEPROM.
3. Build Status – use internal flags to build the driver status word (reference section below).
4. Device Sync – Write fuel gauge context to PS700 RAM.

FIGURE 7-28: END FLOW CHART



7.4.8 BLOCK STATUS

After the end of processing, status information is cleared and a new battery status is obtained, as shown in Figure 7-29 and Figure 7-30. The STATUS bits reside in system RAM.

FIGURE 7-29: STATUS FLOW CHART (PART 1)

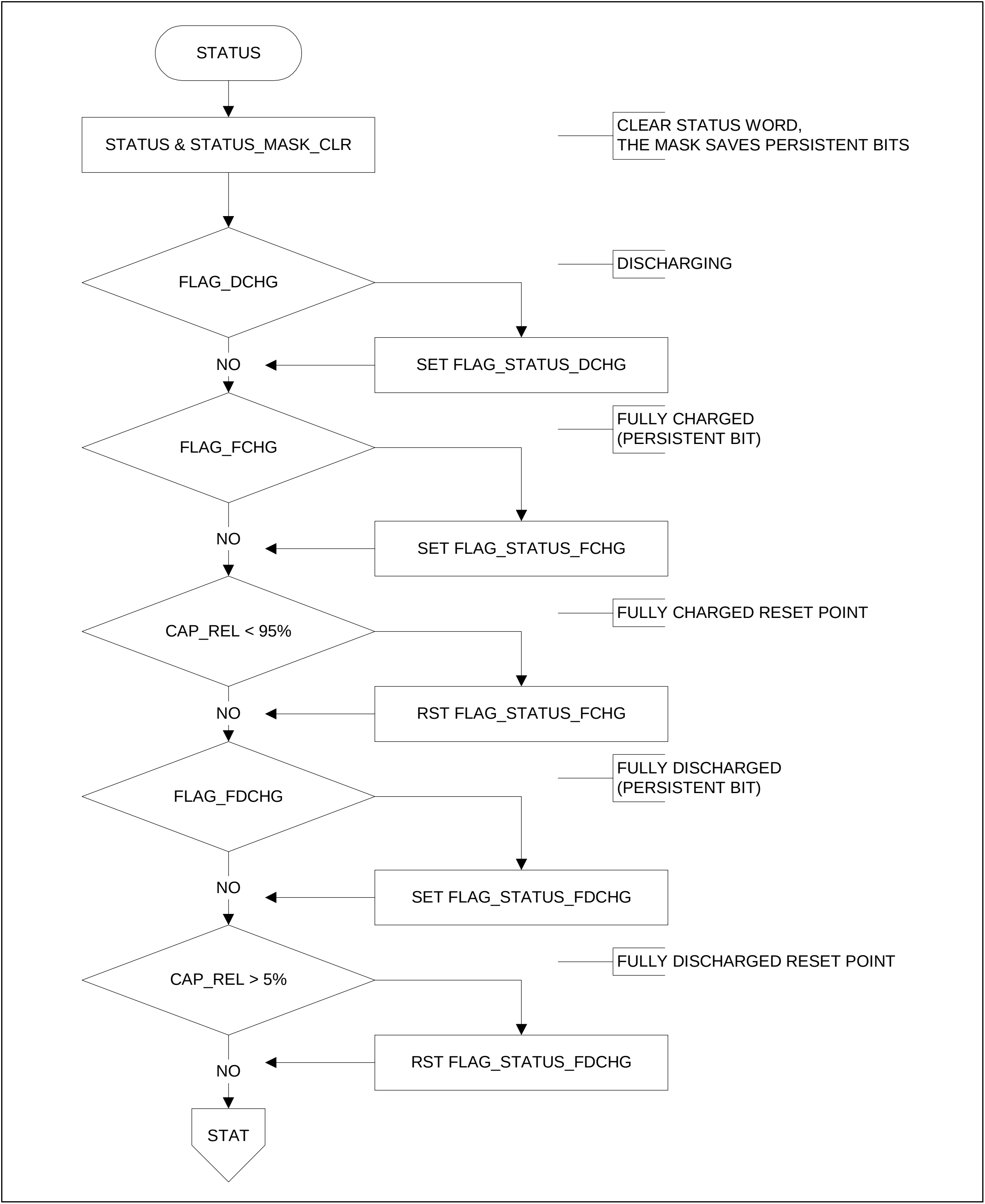
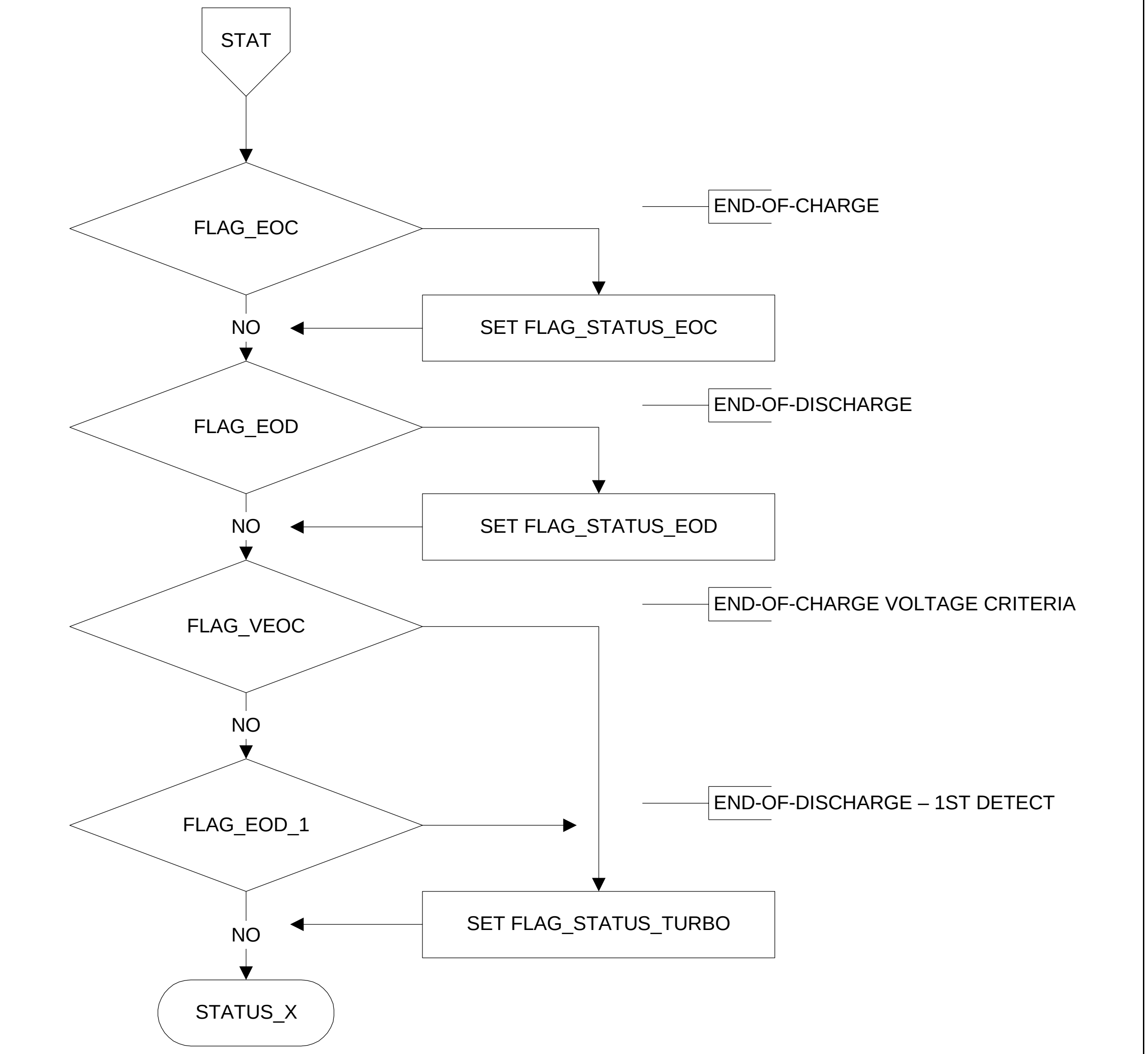


FIGURE 7-30: STATUS FLOW CHART (PART 2)



7.5 Processing Resources

7.5.1 SYSTEM RAM

Table 7-6 shows the processing data stored in system RAM to avoid repeated communication with the PS700 and the corresponding lengths.

TABLE 7-6: SYSTEM RAM PROCESSING RESOURCES

Length		Name	Description
HEX	DEC		
	17	DATA – REPORTED	Processed Fuel Gauge Data
	19	DATA – INTERNAL	Internal Fuel Gauge Data Register
	29	STATIC PARAMETERS	Calibration Factors, Parameters, Configuration
	8	VEOD LUT	VEOD Look-up Table – column axis & 1 row
	14	SCRATCHPAD	Math Accumulators, Command Buffer, etc.
	8	MISC CONTROL REGISTERS	Misc. Control Registers, Flags, etc.
	95	**TOTAL**	

7.5.2 PROGRAM MEMORY

Table 7-7 presents the major fuel gauge functional sections with their corresponding lengths in memory. Functions not appearing in the table are the command decode/jump table and the SMBus communication API. The indented names indicate a subfunction of the name above.

TABLE 7-7: PROGRAM MEMORY PROCESSING RESOURCES

Length		Name	Description
HEX	DEC		
58	88	START-UP	Battery ID, Load Processing Context
6A	106	ADC	Process ADC Results
91	145	CAP	Process Capacity Accumulators
63	99	SDCHG	(option) Self-Discharge
56	86	CAP_CALC	Capacity-based Data (relative cap, etc.)
2E	46	RCAP	(option) Residual Capacity
C8	200	STATE	Charge/Discharge Processing
2C	44	STATE: CHG	Charge Processing – EOC Detect
93	147	STATE: DCHG	Discharge Processing – EOD Detect
58	88	STATE: DCHG: VEOD	(option) VEOD Look-up Table
11	17	STATE: DCHG: VEODAGE	(option) VEOD Aging
3E	62	END	End-Of-Process Housekeeping (status, etc.)
10B	267	UTIL: MATH	Math Utilities and Data Manipulation
6E	110	UTIL: FG	Miscellaneous Fuel Gauge Utilities
48B	**1163**	**TOTAL**	

7.6 Communication

Communication with the PS700 is system dependent and not considered part of the PS700Driver. As implemented on the PowerInfo 2, PS700Driver takes advantage of the available SMBus controller. Generally, the program flow calls for:

1. Loading registers with transaction parameters (i.e., address, byte count, etc.).
2. Building the "script" required by the PowerInfo 2 SMBus controller.
3. Idling to allow the SMBus controller to execute the script.

The PS700Driver follows Figure 7-31 to read from the PS700 and Figure 7-32 to write to it.

FIGURE 7-31: PS700 READ

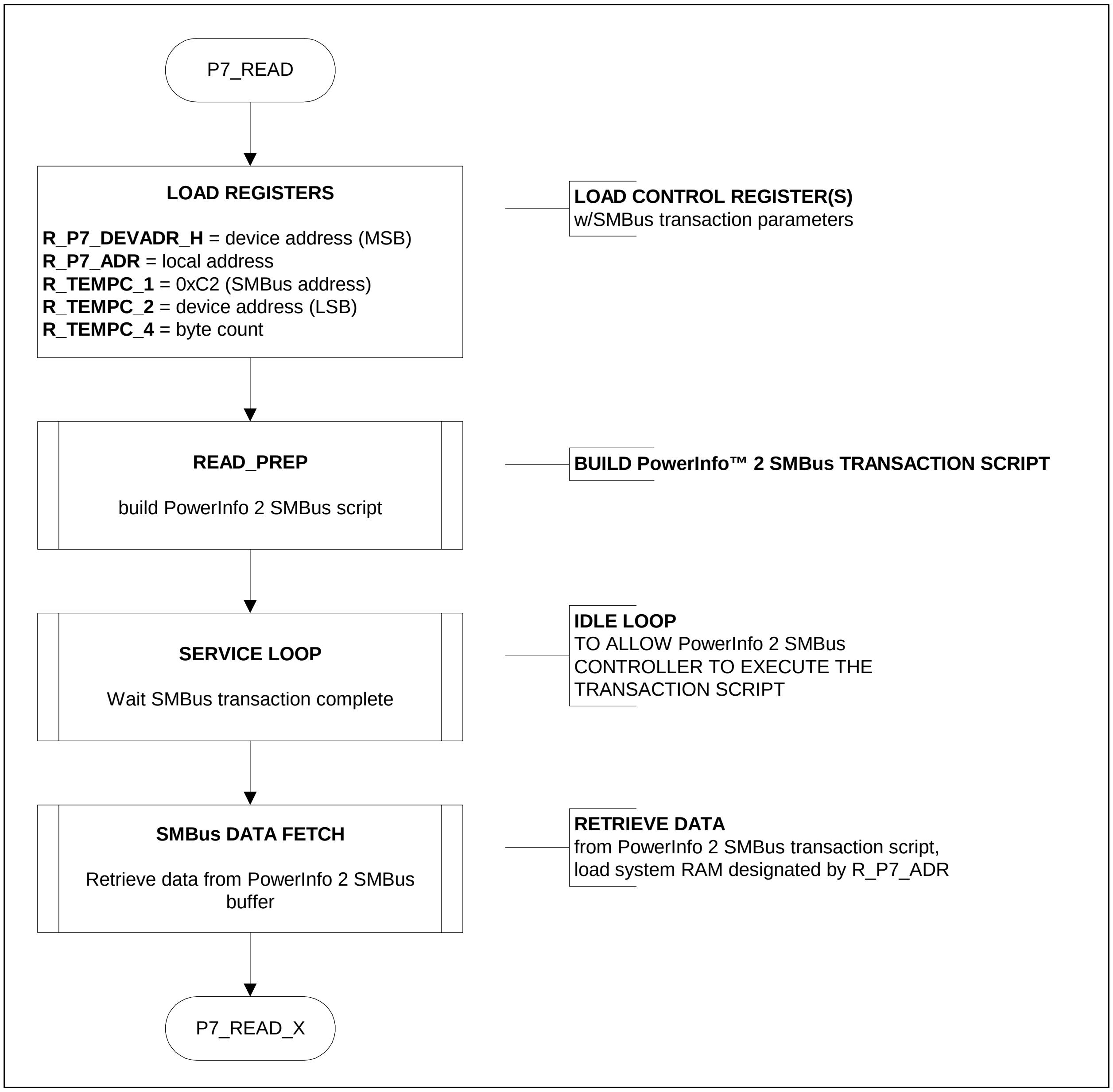
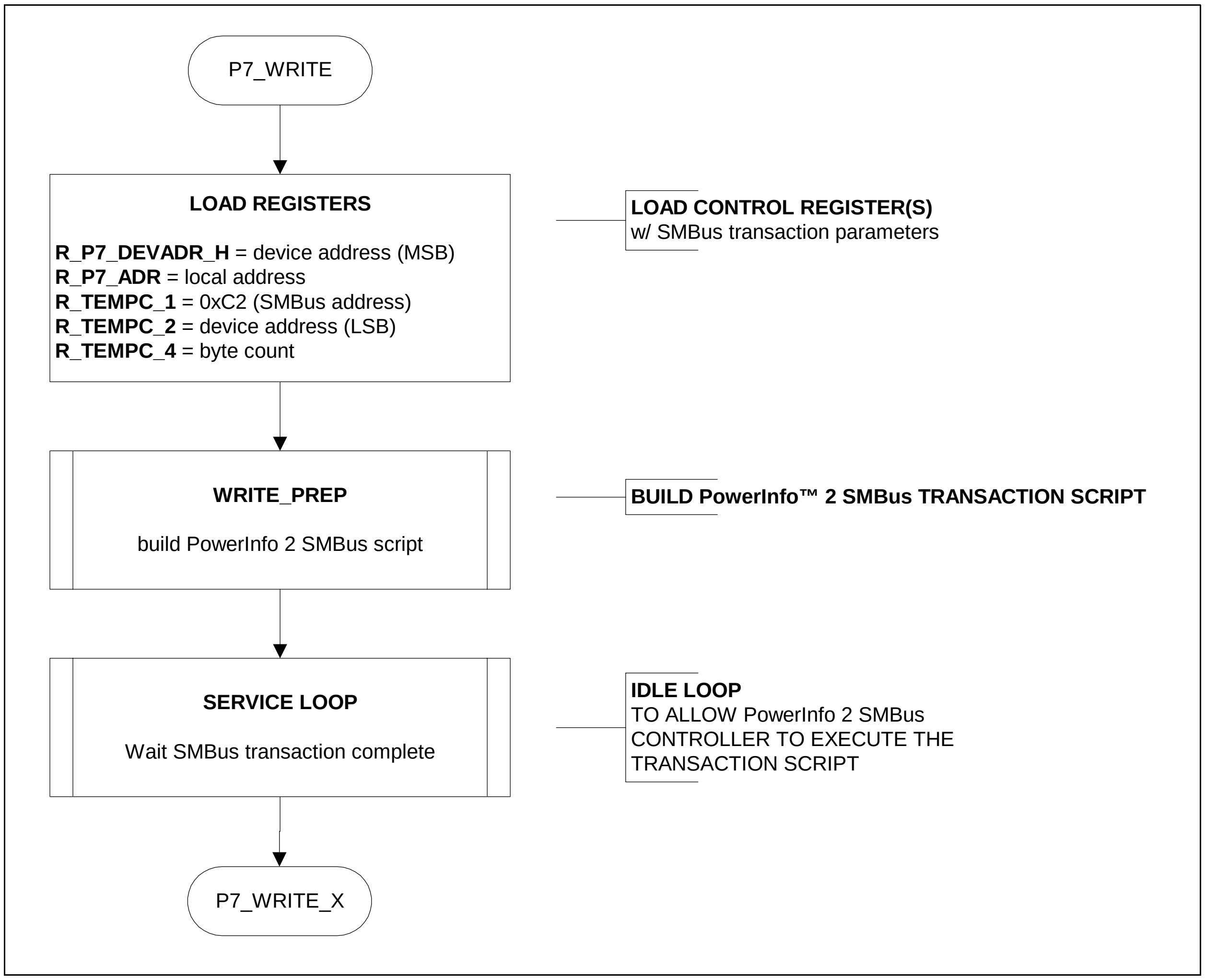

P7_READ
LOAD REGISTERS
R_P7_DEVADR_H = device address (MSB)
R_P7_ADR = local address
R_TEMPC_1 = 0xC2 (SMBus address)
R_TEMPC_2 = device address (LSB)
R_TEMPC_4 = byte count
LOAD CONTROL REGISTER(S) w/SMBus transaction parameters
READ_PREP
build PowerInfo 2 SMBus script
BUILD PowerInfo™ 2 SMBus TRANSACTION SCRIPT
SERVICE LOOP
Wait SMBus transaction complete
IDLE LOOP TO ALLOW PowerInfo 2 SMBus CONTROLLER TO EXECUTE THE TRANSACTION SCRIPT
SMBus DATA FETCH
Retrieve data from PowerInfo 2 SMBus buffer
RETRIEVE DATA from PowerInfo 2 SMBus transaction script, load system RAM designated by R_P7_ADR
P7_READ_X

FIGURE 7-32: PS700 WRITE


P7_WRITE
LOAD REGISTERS
R_P7_DEVADR_H = device address (MSB)
R_P7_ADR = local address
R_TEMPC_1 = 0xC2 (SMBus address)
R_TEMPC_2 = device address (LSB)
R_TEMPC_4 = byte count
LOAD CONTROL REGISTER(S) w/ SMBus transaction parameters
WRITE_PREP
build PowerInfo 2 SMBus script
BUILD PowerInfo™ 2 SMBus TRANSACTION SCRIPT
SERVICE LOOP
Wait SMBus transaction complete
IDLE LOOP TO ALLOW PowerInfo 2 SMBus CONTROLLER TO EXECUTE THE TRANSACTION SCRIPT
P7_WRITE_X

8.0 PS700 CONFIGURATION

The PS700Driver assumes that the Accumulator Control register (Accumctrl) is configured as shown in Table 8-1 and the A/D Control registers (ADcx) are configured as shown in Table 8-2.

TABLE 8-1: ACCUMULATOR CONTROL REGISTER

	Name	Enable	Description
0	(RES)	0	Reserved
1	(RES)	0	Reserved
2	(RES)	0	Reserved
3	TSEL	1/0	Temperature Select
4	ACCV	0	Accumulate Voltage
5	ACCT	1	Temperature (temperature time used for self discharge)
6	ACCL	1	Current Accumulation
7	ACCUM	1	Master Enable

TABLE 8-2: A/D CONTROL REGISTERS

	Name	Enable	Description
0	Ires	1	Current
1	ITres	1	Internal Temperature
2	ETres	0	External Temperature
3	VPres	1	Pack Voltage
4	VC1ENTres	1	VCELL1 Voltage
5	VC2res	1	VCELL2 Voltage
6	OFFSres	0	Offset
7	AUXres	0	Aux

NOTES:

MICROCHIP®

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: www.microchip.com

Atlanta
3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034
Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848
Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
16200 Addison Road, Suite 255
Addison Plaza
Addison, TX 75001
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, IN 46902
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
25950 Acero St., Suite 200
Mission Viejo, CA 92691
Tel: 949-462-9523
Fax: 949-462-9608

San Jose
1300 Terra Bella Avenue
Mountain View, CA 94043
Tel: 650-215-1444
Fax: 650-961-0286

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Unit 32 41 Rawson Street
Epping 2121, NSW
Sydney, Australia
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Unit 706B
Wan Tai Bei Hai Bldg.
No. 6 Chaoyangmen Bei Str.
Beijing, 100027, China
Tel: 86-10-85282100
Fax: 86-10-85282104

China - Chengdu
Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200
Fax: 86-28-86766599

China - Fuzhou
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506
Fax: 86-591-7503521

China - Hong Kong SAR
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700
Fax: 86-21-6275-5060

China - Shenzhen
Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380
Fax: 86-755-8295-1393

China - Shunde
Room 401, Hongjian Building, No. 2
Fengxiangnan Road, Ronggui Town, Shunde
District, Foshan City, Guangdong 528303, China
Tel: 86-757-28395507 Fax: 86-757-28395571

China - Qingdao
Rm. B505A, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-22290061 Fax: 91-80-22290062

Japan
Yusen Shin Yokohama Building 10F
3-17-2, Shin Yokohama, Kohoku-ku,
Yokohama, Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5932 or
82-2-558-5934

Singapore
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Kaohsiung Branch
30F - 1 No. 8
Min Chuan 2nd Road
Kaohsiung 806, Taiwan
Tel: 886-7-536-4816
Fax: 886-7-536-4817

Taiwan
Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

Taiwan
Taiwan Branch
13F-3, No. 295, Sec. 2, Kung Fu Road
Hsinchu City 300, Taiwan
Tel: 886-3-572-9526
Fax: 886-3-572-6459

EUROPE

Austria
Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45-4420-9895 Fax: 45-4420-9910

France
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany
Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy
Via Salvatore Quasimodo, 12
20025 Legnano (MI)
Milan, Italy
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands
Waegenburghtplein 4
NL-5152 JR, Drunen, Netherlands
Tel: 31-416-690399
Fax: 31-416-690340

United Kingdom
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44-118-921-5869
Fax: 44-118-921-5820

07/12/04