Application Note

DA9063-A Power Management for R-Car H3 Platform

AN-PM-085

Abstract

The R-Car H3 System-on-Chip (SoC)-based platform from Renesas is part of a family of platforms (R-Car series) for automotive infotainment systems. The H3 is aimed at the high-end segment, and is optimized for automotive Human Machine Interface (HMI), infotainment and integrated dashboards.

The platform features the Dialog DA9063-A as system PMIC (Power Management IC) and the Dialog DA9213-A and DA9214-A multi-phase sub-PMIC step-down buck converters to power and supervize the complete system.

Through a description of the general system configuration, power capabilities and requirements and an overview of the component interconnections, it will be shown that the combination of DA9063-A, DA9213-A, and DA9214-A are highly suited as the R-Car power management system solution for H3 platforms.

DA9063-A Power Management for R-Car H3 Platform

Contents

Figures

Tables

DA9063-A Power Management for R-Car H3 Platform

1 Introduction

This document describes how to interconnect the DA9063-A Power Management IC (PMIC) and the DA9213-A and DA9214-A sub-PMICs to the Renesas R-Car H3 System on a Chip (SoC). The DA9063-A is a highly integrated chip that supports Dynamic Voltage Control (DVC) technology, enabling significant power saving: this feature supports the Dynamic Voltage and Frequency Scaling (DVFS) technology that is used by many processors.

As a result of their highly integrated features, the DA9063-A PMIC, DA9213-A, and DA9214-A sub-PMICs significantly reduce the overall system cost and size compared to a discrete solution. This application note addresses only the power supply related features: discussion of other features of the optimized PMIC is beyond the scope of this document.

For further information on the DA9063-A, DA9213-A, and DA9214-A please refer to the datasheets available via your local Dialog sales office.

For information about Renesas R-Car H3 SoC, please refer to Renesas website:

https://www.renesas.com/en-us/solutions/automotive/products/rcar-h3.html

2 Renesas R-Car H3 SoC Description

Renesas R-Car H3 is a platform for automotive infotainment with an SoC containing ten cores (ARM®Cortex®-A57 Quad Core, ARM Cortex-A53 Quad, ARM Cortex-R7 Dual lock-step) and PowerVR GX6650 GPU.

Figure 1 shows a typical system block diagram of the R-Car H3 SoC application. The embedded cores require suitable power management that is readily achieved using the Dialog DA9063-A, DA9213-A, and DA9214-A.

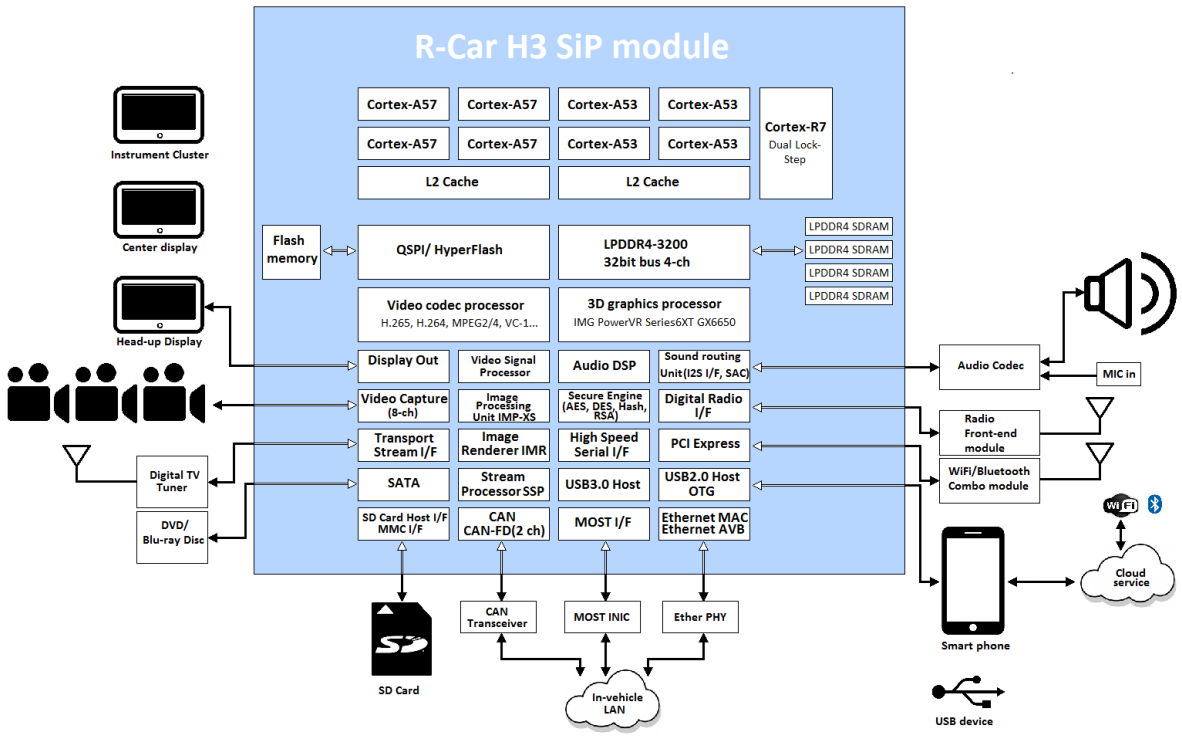

Figure 1: R-Car H3 System Block Diagram

3 DA9063-A, DA9213-A, and DA9214-A Description

The DA9063-A (Figure 2) is a high-current system PMIC suitable for dual- and quad-core processors that require up to 5 A core processor supply. The DA9063-A contains:

- Six DC-DC buck converters designed to use small external 1 µH inductors, capable of supplying in total up to 12 A continuous output current (0.3 V to 3.3 V). The buck converters do not require external Schottky diodes; they dynamically optimize their efficiency depending on the load-current using an Automatic Sleep Mode (ASM) and incorporate pin and software controlled Dynamic Voltage Control (DVC) to support processor load adaptive adjustment of the supply voltage. In addition BuckPro includes the facility to implement VTT memory bus termination if required.
- 11 SmartMirror™ programmable low-dropout (LDO) regulators rated up to 300 mA. All support remote capacitor placement and can operate from low 1.5 V/1.8 V input supplies. This allows these LDOs to be cascaded with (in other words: supplied by) a suitable buck supply to improve overall system efficiency.

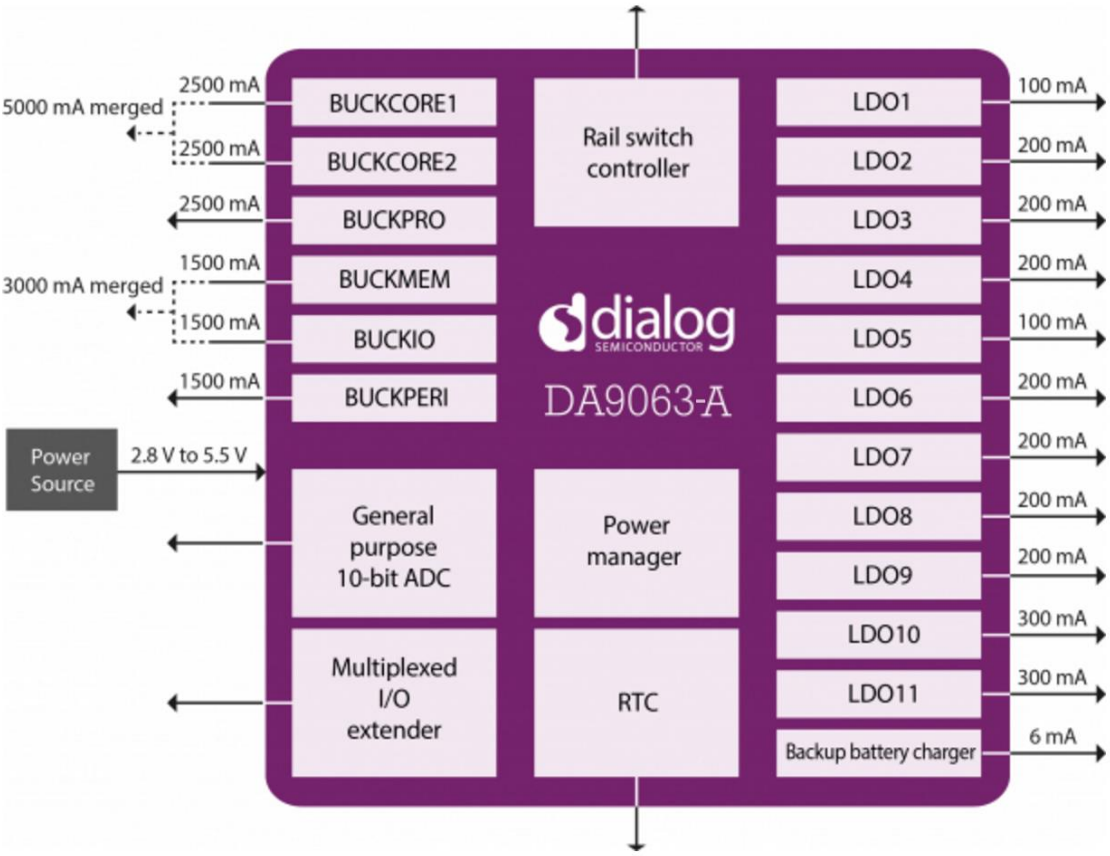

Figure 2: DA9063-A System Block Diagram

DA9063-A Power Management for R-Car H3 Platform

The DA9213-A (Figure 3) and DA9214-A (Figure 4) are multi-phase, single- and dual-output, synchronous step-down converters suitable for supplying CPUs that require high currents. Each converter operates using a small external 0.22 µH inductor on each phase. They produce an output voltage in the range of 0.3 V to 1.57 V. The input voltage range of 2.8 V to 5.5 V makes them suited to a wide variety of low-voltage systems.

To guarantee the highest accuracy and support multiple PCB routing scenarios without loss of performance, a remote sensing capability is implemented on each DA9213-A and DA9214-A output.

The DA9213-A buck operates with four phases and is capable of delivering up to 20 A continuous output current.

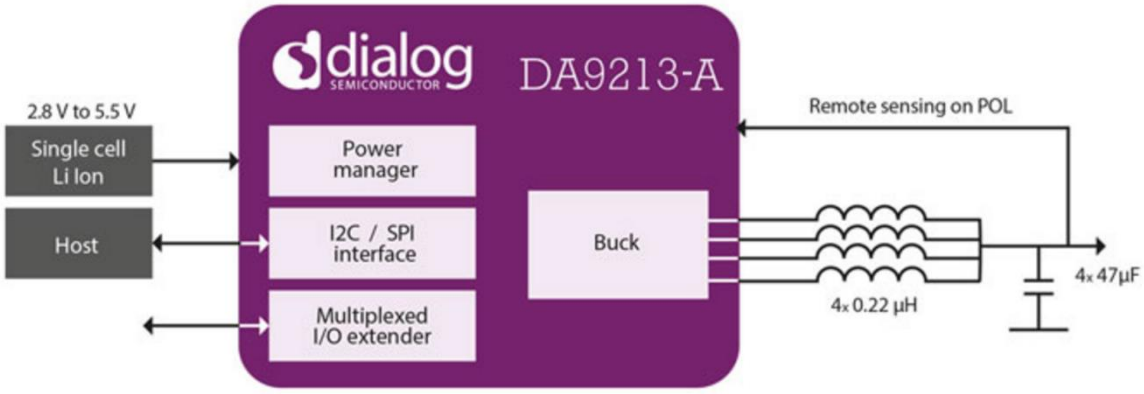

Figure 3: DA9213-A System Block Diagram

Each DA9214-A buck operates with two phases and is capable of delivering up to 10 A continuous output current per buck.

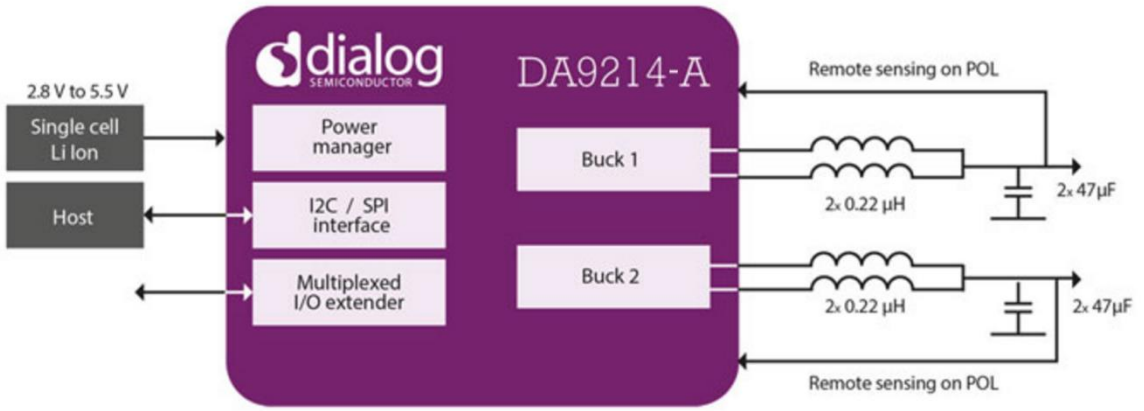

Figure 4: DA9214-A System Block Diagram

4 R-Car H3 SoC Power Requirements

Several power domains in the R-Car H3 SoC platform require precise voltage management for reliable system operation. The primary power domains are:

- DFVS_0.8V
- VDD_0.8V
- DDR_1.1V
- DDR_1.8V

Other supplies will be required for peripherals, I/O interfaces, SD cards, and such. Additionally, the system power management must comply with the specific power-up and power-down sequence guidelines for the R-Car SoC (shown in Figure 5).

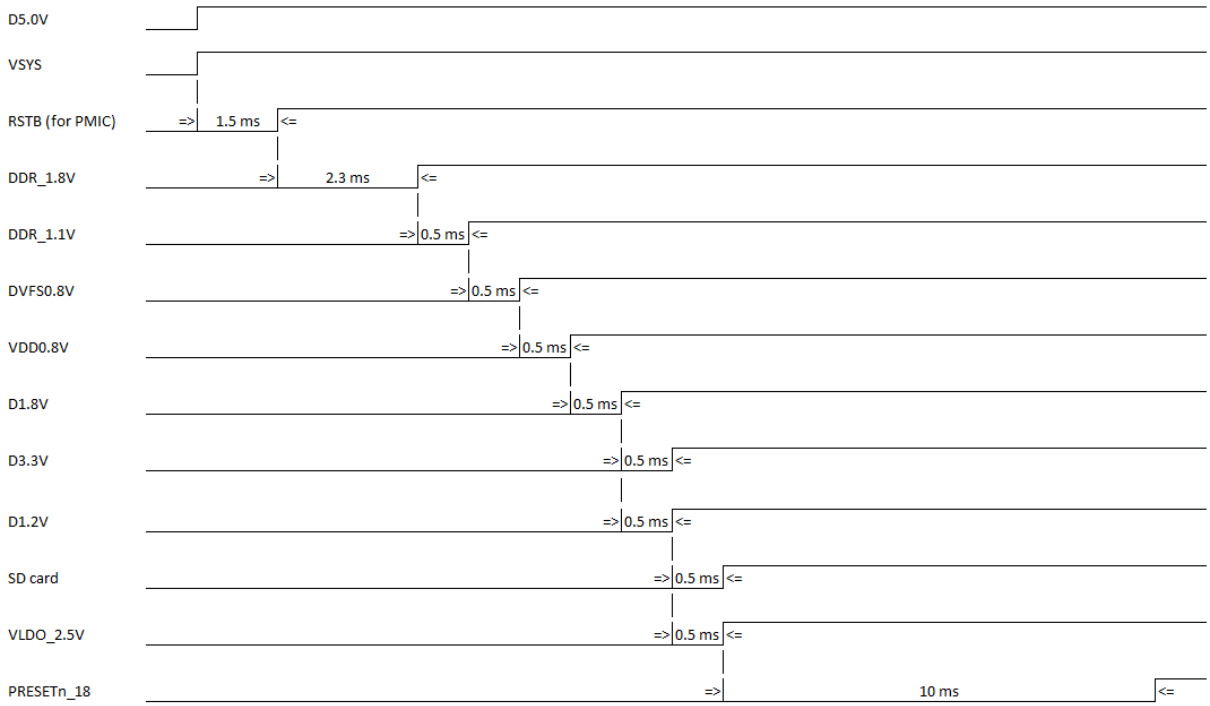

Figure 5: Start-Up Sequence (Timing Is Not To Scale)

4.1 Memory Retention Mode

The R-Car H3 SoC implements a mode whereby the power supplies to the DDR memory are maintained during memory retention mode to preserve the memory contents. During memory retention the load current taken by the DDR memory is very low and all other supplies are disabled.

Figure 6 shows the PMIC interconnections for a system where all the DDR memory is powered as a single block. If the application necessitates splitting the memory architecture into two blocks this is shown in Figure 7. The second memory block can be enabled or disabled by the SoC enabling or disabling the rail switch controllers. In memory retention mode the second memory block will be powered down.

5 R-Car H3 SoC Power Tree System Diagram

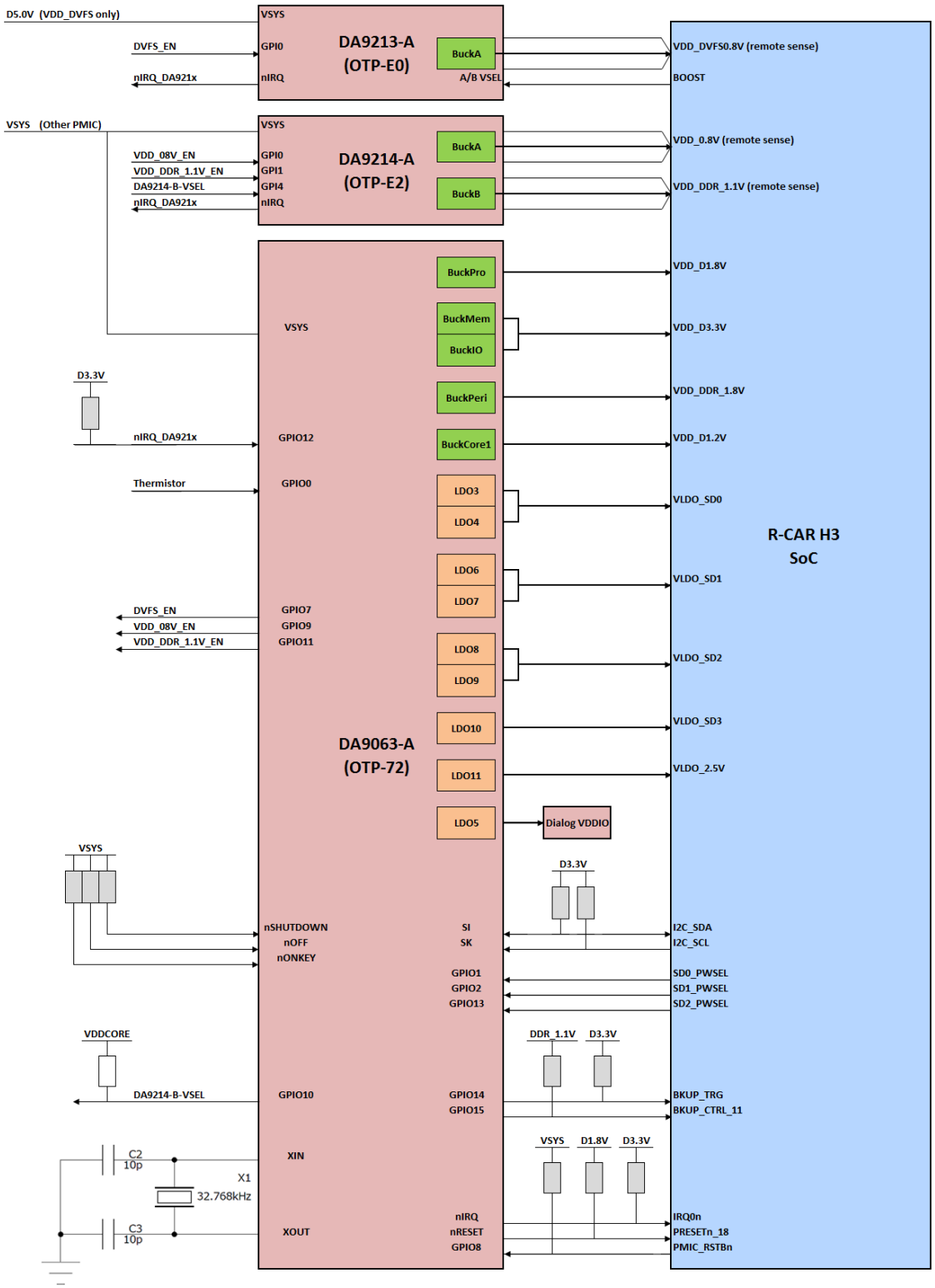

Figure 6: R-Car H3 and PMIC Interconnections

6 R-Car H3 SoC Power Tree System Diagram (Split Memory)

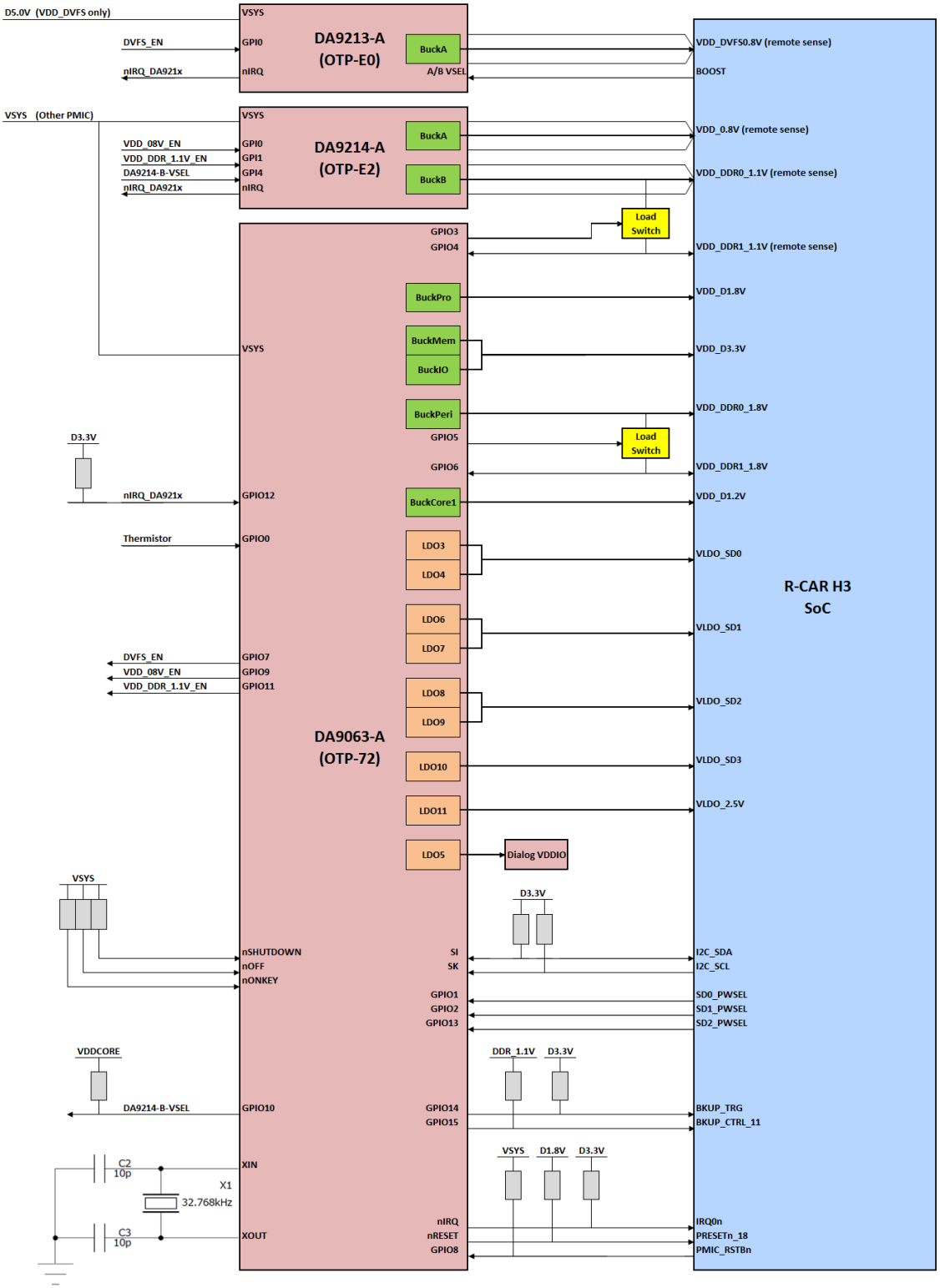

Figure 7: R-Car H3 and PMIC Interconnections (Split Memory)

7 Cold boot Sequence for R-Car H3

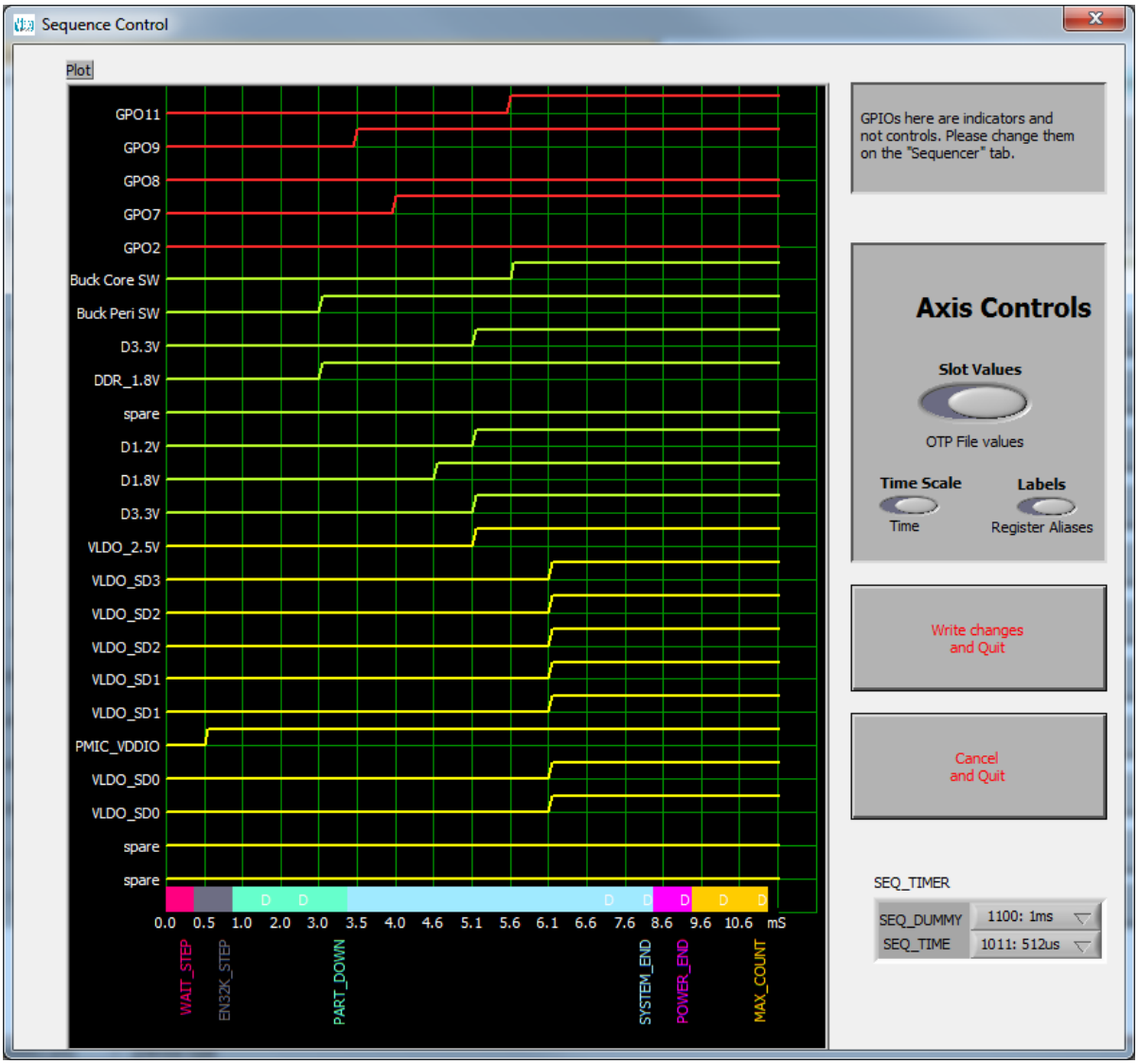

Figure 8: DA9063-72-A Power-Up Sequence

Please contact your local Dialog representative for the recommended OTPs.

8 Operation

When 5 V is applied to the D5.0V and V_{SYS} supplies the DA9063-A system PMIC starts up automatically. It follows the start-up sequence programmed in the OTP enabling output power rails in the order specified. GPIO 7, 9, and 11 are configured to control the enabling of the three sub-PMIC bucks and are also part of the power sequencer timing. In this way the start-up timing of the sub-PMIC buck outputs are controlled.

Once the sequencer has completed the start-up sequence the nRESET signal from the DA9063-A is released to allow the SoC to start operation.

The outputs of LDO3 and LDO4 are combined to provide the specified 300 mA load current for the SD0 card supply. Similarly, LDO6 and LDO7 are combined as are LDO8 and LDO9 to power SD1 and SD2 cards respectively. LDO10 is a 300 mA LDO and can supply the SD3 card individually. If fewer SD cards are used in a customer end application or lower current SD cards are used then unused LDOs may be reused elsewhere in the end application.

GPIO1, 2, and 13 provide the ability for the SoC to select the SD0-2 card output voltage by controlling the logic level applied to the GPIO input. A logic low from the SoC produces an SD card voltage of 1.8 V output on the respective output; a logic high produces an output of 3.3 V. The SD3 card voltage is set in OTP to be 1.8 V with the ability to change this to 3.3 V via an I^2C write.

LDO5 is used to generate the internal power supplies for the Dialog system and sub-PMICs.

8.1 DVFS and AVS

The DVFS voltage is set in OTP to be the default start-up voltage that is guaranteed to ensure the system powers up and runs. Deviations in the manufacturing process result in some processors that are capable of operating from lower voltages than the nominal. AVS allows for adjustments to the DVFS voltage to cater for these processors.

The SoC can adjust the DVFS set voltage by an I^2C write to the DA9213-A, VBUCKA_CTRL_A register (address 0xD7). The output voltage can be adjusted in 10 mV steps.

The DVFS power rail can also be switched to a higher voltage for a short period of time. The higher voltage, BOOST mode is selected when the SoC takes the BOOST pin high.

The BOOST voltage is set in OTP to be the default boost voltage and can be adjusted by an I^2C write to the DA9213-A, VBUCKA_CTRL_B register (address 0xD8). The output voltage can be adjusted in 10 mV steps.

8.2 Memory Retention Mode (Sleep mode)

PMIC_RSTBn from the SoC is used to enter and exit memory retention mode. When PMIC_RSTBn is taken to a logic low the system PMIC performs a power-down sequence. During the power-down sequence the bucks supplying the DDR1.1V and DDR1.8V outputs are re-configured to prevent them from switching off.

Under normal ACTIVE mode conditions all bucks are programmed to operate in PWM mode to produce predictable noise performance. Efficiency is reduced at low load currents when operating in this mode so when entering memory retention mode the bucks supplying DDR1.1V and DDR1.8V are automatically changed to operate in Pulse-Frequency Modulation (PFM) mode. In doing this the quiescent current from the 5 V input is reduced to less than 1 mA. If a split memory architecture is implemented then only Bank0 will be preserved during memory retention mode, Bank1 will be turned off.

When PMIC_RSTBn is taken to a logic high once more the DDR1.1V and DDR1.8V bucks are automatically re-configured to operate in Pulse-Width Modulation (PWM) mode before the system returns fully to ACTIVE mode by following the start-up sequence.

In order to enter memory retention mode the SoC first places the DDR memory in self-refresh mode to ensure the contents are retained.

8.2.1 Warm Boot vs Cold Boot

Figure 10 shows the procedure for exiting from memory retention mode. Memory retention mode exit is triggered by a wake-up event. A wake-up event can be triggered from a wake-up enabled GPIO edge, nONKEY going low, the SYS_EN pin rising or an RTC alarm being triggered. The wake-up event causes the PMIC to move up the sequencer and then release nRESET to start the SoC. At this point the SoC will begin the software boot-up procedure.

During system boot-up the SoC checks the state of BKUP_TRG to determine if the start-up requires a cold or warm boot process. BKUP_TRG is programmed in OTP to be low during a system power-up to indicate a cold boot is required. Before entering memory retention mode the SoC sets BKUP_TRG to be high to indicate a warm boot is required when the system is next started. If all power is lost BKUP_TRG reverts to the OTP setting which results in a cold boot.

If the SoC GPIO, GP1-08, is connected to BKUP_TRG output pin on DA9063-A the SoC can directly determine the BKUP_TRG status by reading GP1-08.

If GP1-08 is not connected to BKUP_TRG the SoC can determine if a warm or cold boot is required by either reading BKUP_TRG status via an I^2C read of the DA9063-A or an I^2C read of GP_ID_1 (address 0x122) register in DA9063-A. This register is programmed to be 0x00 in the OTP but the register value can be changed by the SoC using an I^2C write. The register value is sustained through entry and exit of memory retention mode but reset to OTP value during power-up from off. The SoC can set GP_ID_1 to a non-zero value before entering memory retention mode and when booting up can interrogate the register to determine whether the boot-up is warm or cold.

Please refer to 0 for examples of the software implementation.

8.2.2 BKUP_CTRL

BKUP_CTRL is connected to DA9063 GPIO4. The SoC sets BKUP_CTRL when entering memory retention mode and clears it when exiting. Please refer to 0 for examples of the software implementation.

8.2.3 Sleep Timer

Taking PMIC_RSTBn high will result in a system wake-up event, with the PMIC following the power-up sequence.

An alternative approach is to use a sleep timer to wake the system after a predetermined time in memory retention mode.

This can be implemented on the Dialog power management solution by use of the RTC clock and RTC alarm function.

As before the SoC sets BKUP_TRG to be a logic high to indicate a warm boot when exiting sleep. Before entering sleep mode the SoC reads the RTC time within the DA9063-A. The SoC adds the required sleep interval to the RTC time and sets the DA9063-A alarm time to this new value. The SoC then sets the alarm to be active. Finally the SoC clears all interrupts before entering memory retention mode.

After the set time has elapsed the RTC alarm is triggered causing a wake-up event to be triggered within the DA9063-A. This causes the PMIC to move from memory retention mode to active waking up the SoC in the process.

The SoC checks BKUP_ TRG state and determines the wake-up event is a warm boot.

Figure 9 shows the process by which the SoC enters memory retention mode if required. Firstly the SoC determines whether the system is shutting down to off or memory retention mode. If memory retention mode is required the following sequence is followed:

- The SoC performs an I^2C write of 0x99 to PMIC register address 0x94.
- The SoC performs an I^2C write to set PMIC GPIO3 = 1
  - An alternative option is the SoC performs an I^2C write to set GP_ID_1 = 0x01

- If a wake from memory retention mode after a predetermined time is required the SoC performs an I²C read of the RTC time, adds on the required sleep time and writes back an RTC ALARM time into the PMIC before setting the RTC ALARM_ON bit.
- The SoC sets the DDR into self-refresh mode before performing an I²C write of 0x02 to register address 0x0E, thus clearing the SYSTEM_EN bit.

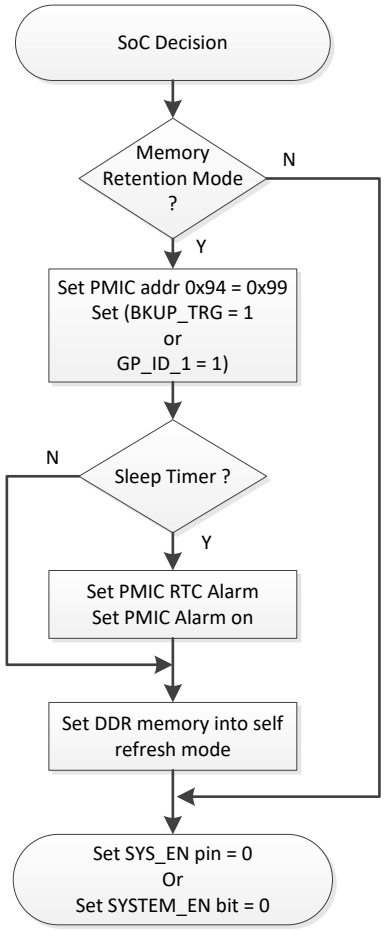

Figure 9: SoC Sequence for Memory Retention Entry

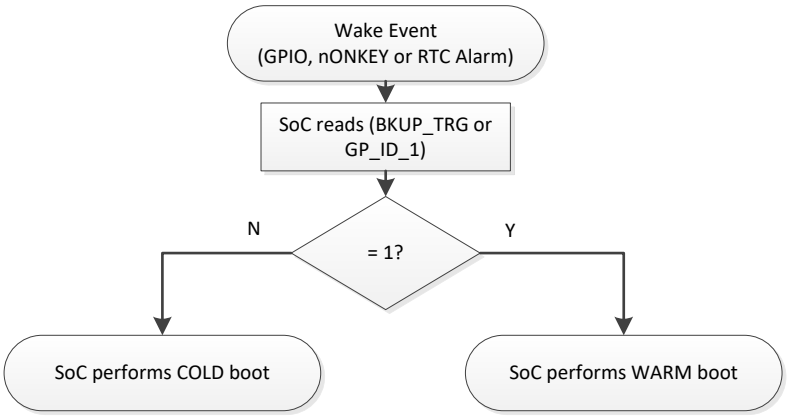

Figure 10: SoC Sequence for Memory Retention Exit

Please refer to 0 for examples of the software implementation.

9 Reference Design

Figure 11 shows a photograph of a Dialog R-Car H3 reference design evaluation board PCB. This PCB allows the PMIC solution to be evaluated. The 42 mm x 30 mm white lined box in the center of the PCB shows the core, active area used for components and tracking of the solution. The remainder of the PCB is purely for easy access connections.

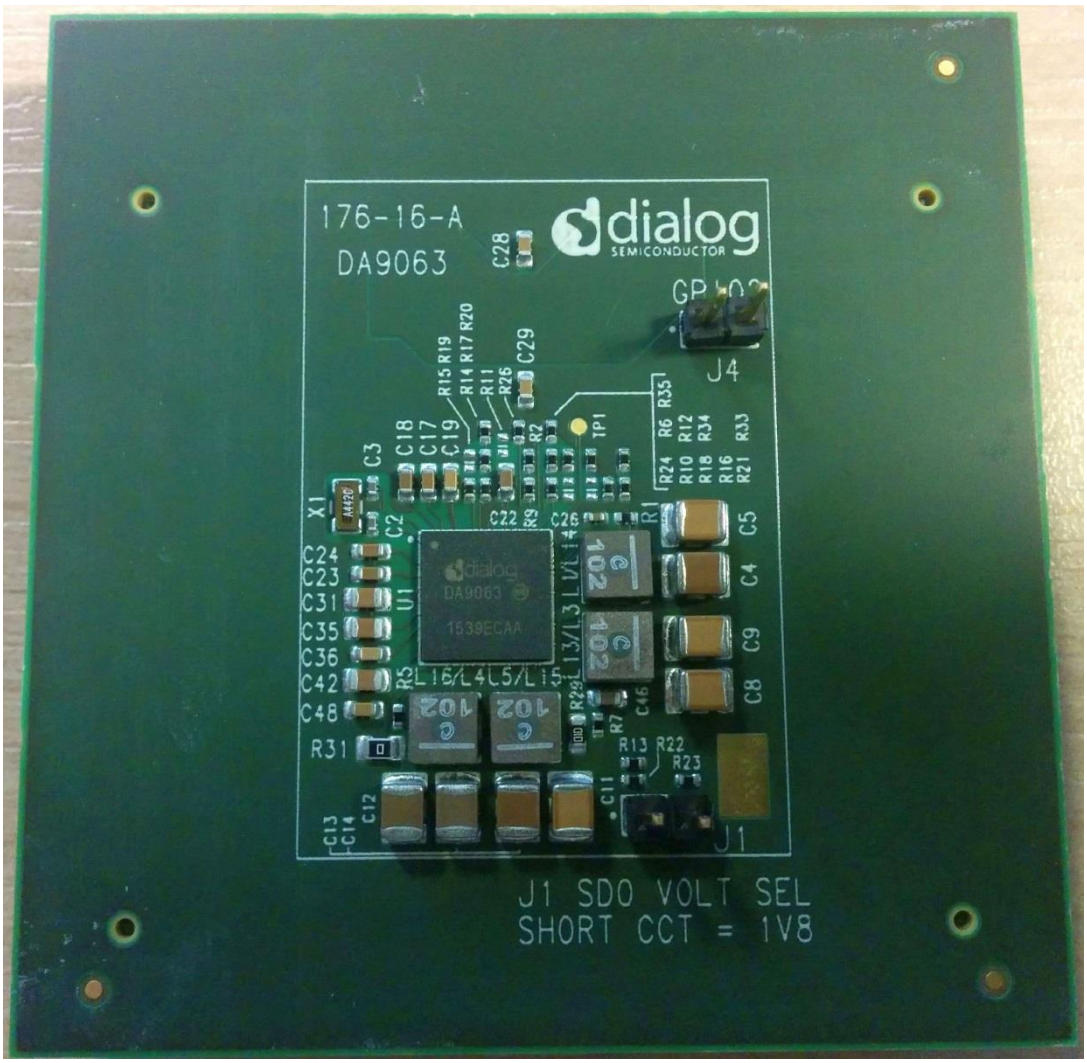

Figure 11: The Dialog R-Car H3 Reference Design

9.1 Measurement Results

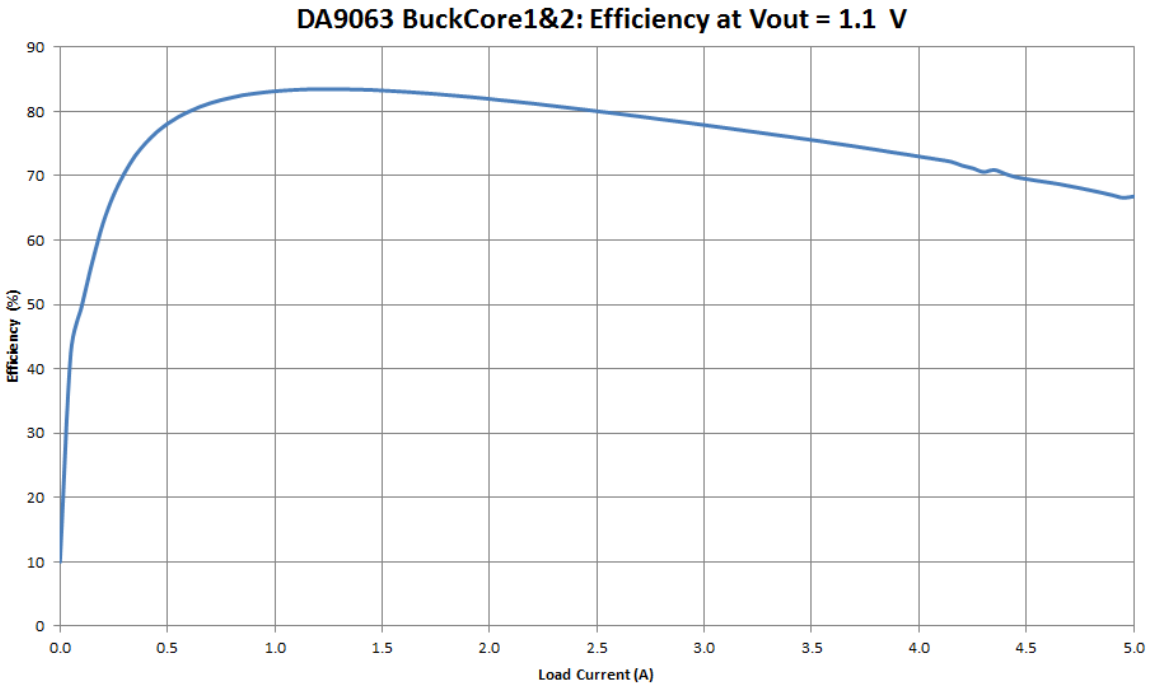

Figure 12: DDR_1.1V Efficiency

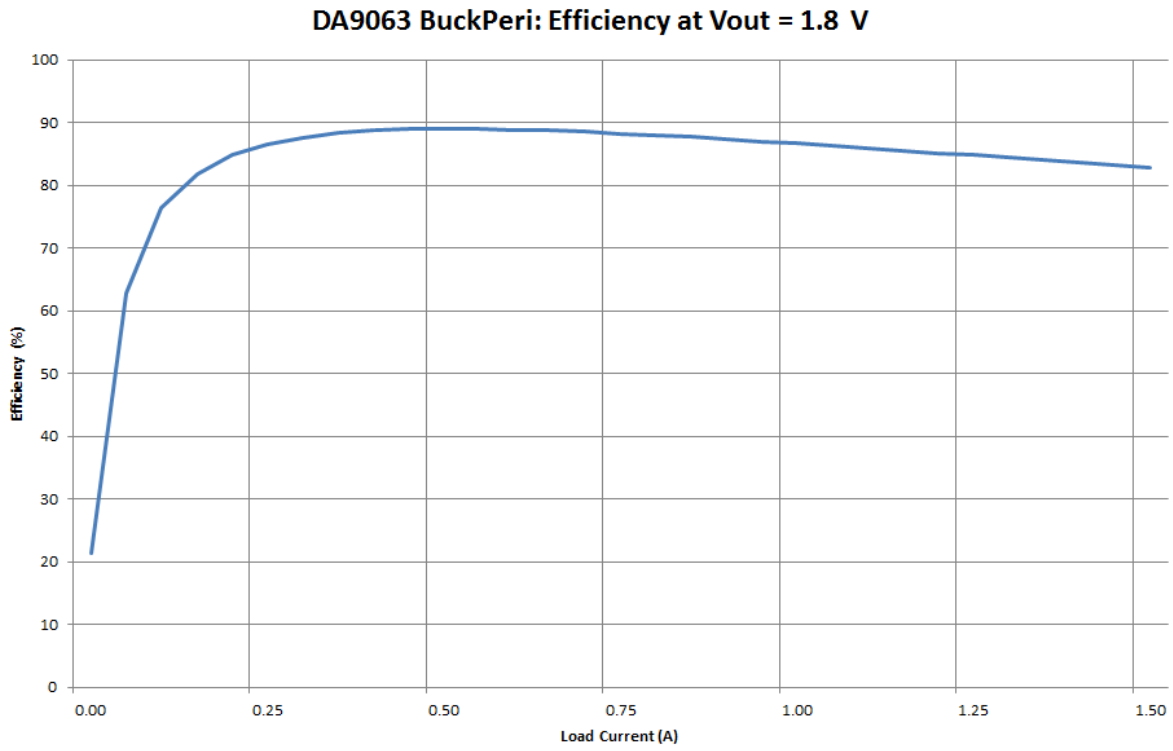

Figure 13: DDR_1.8V Efficiency

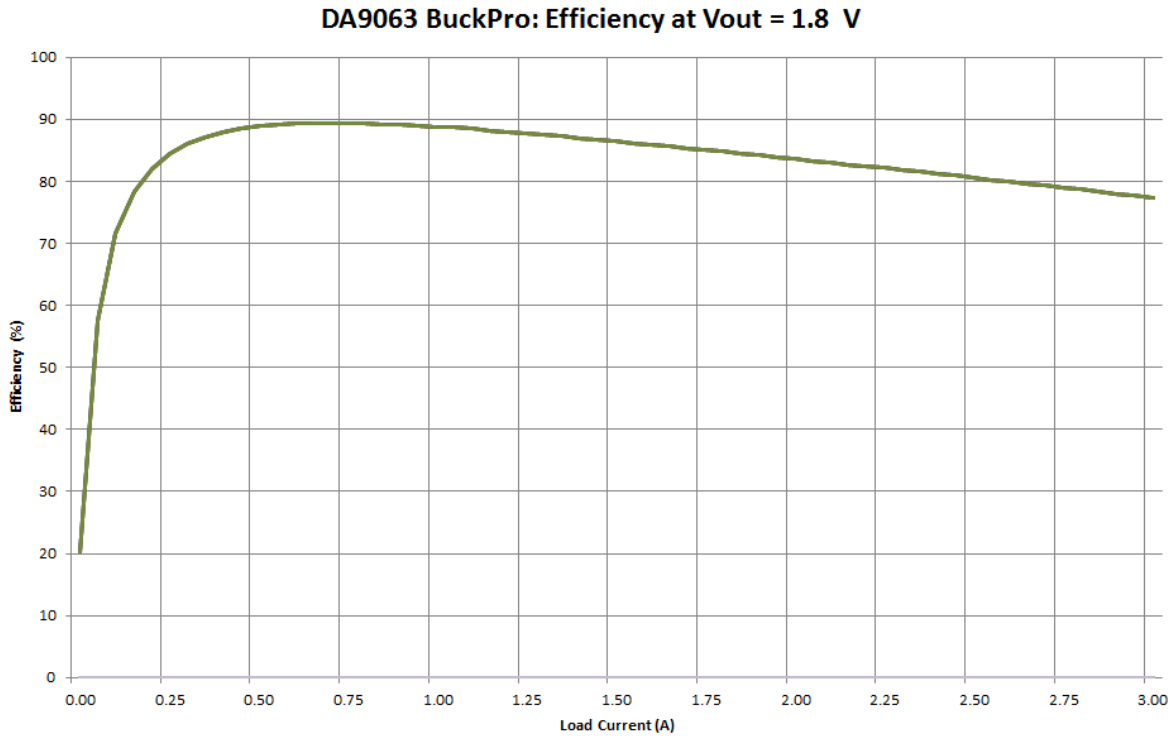

Figure 14: D1.8V Efficiency

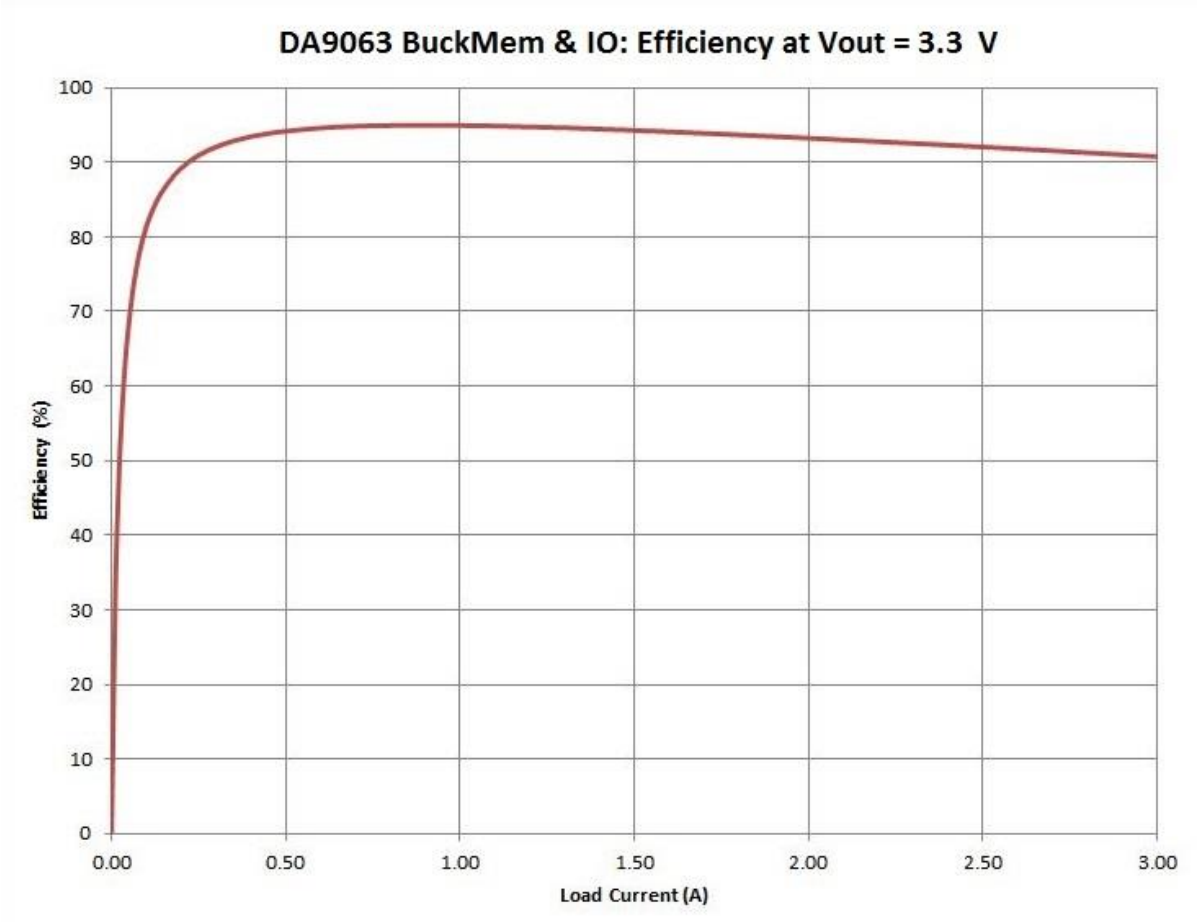

Figure 15: D3.3V Efficiency

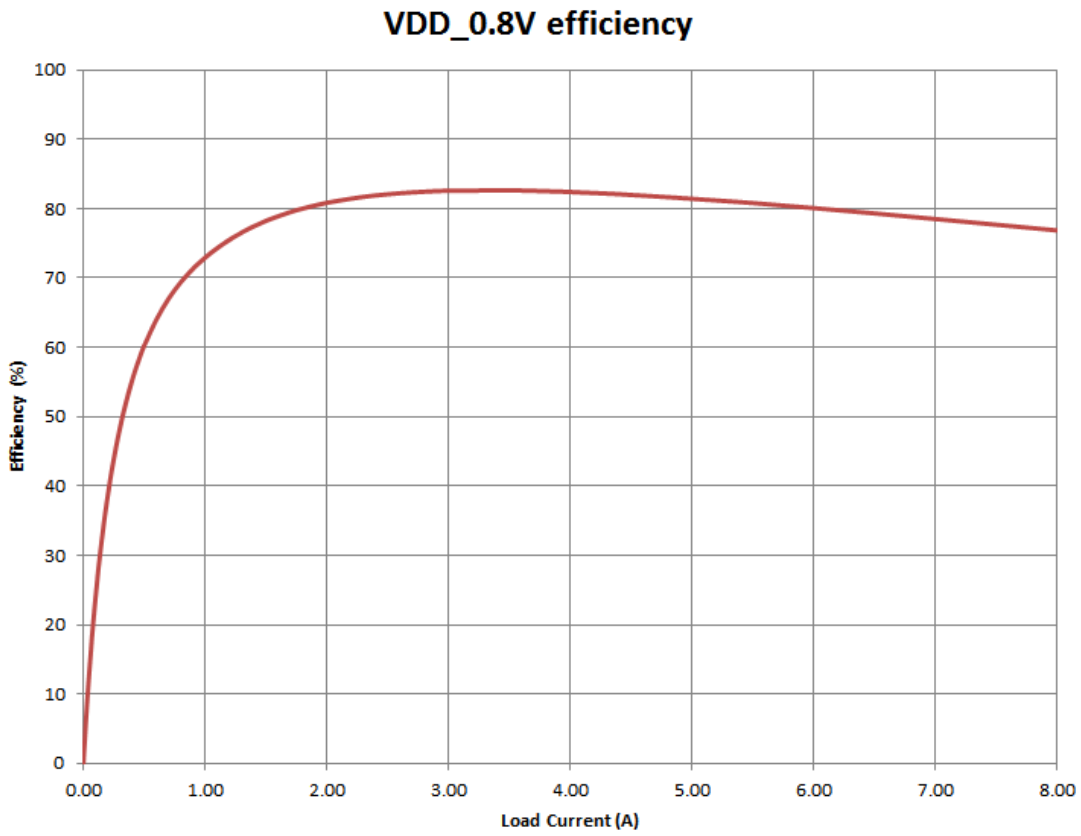

Figure 16: VDD_08 Efficiency

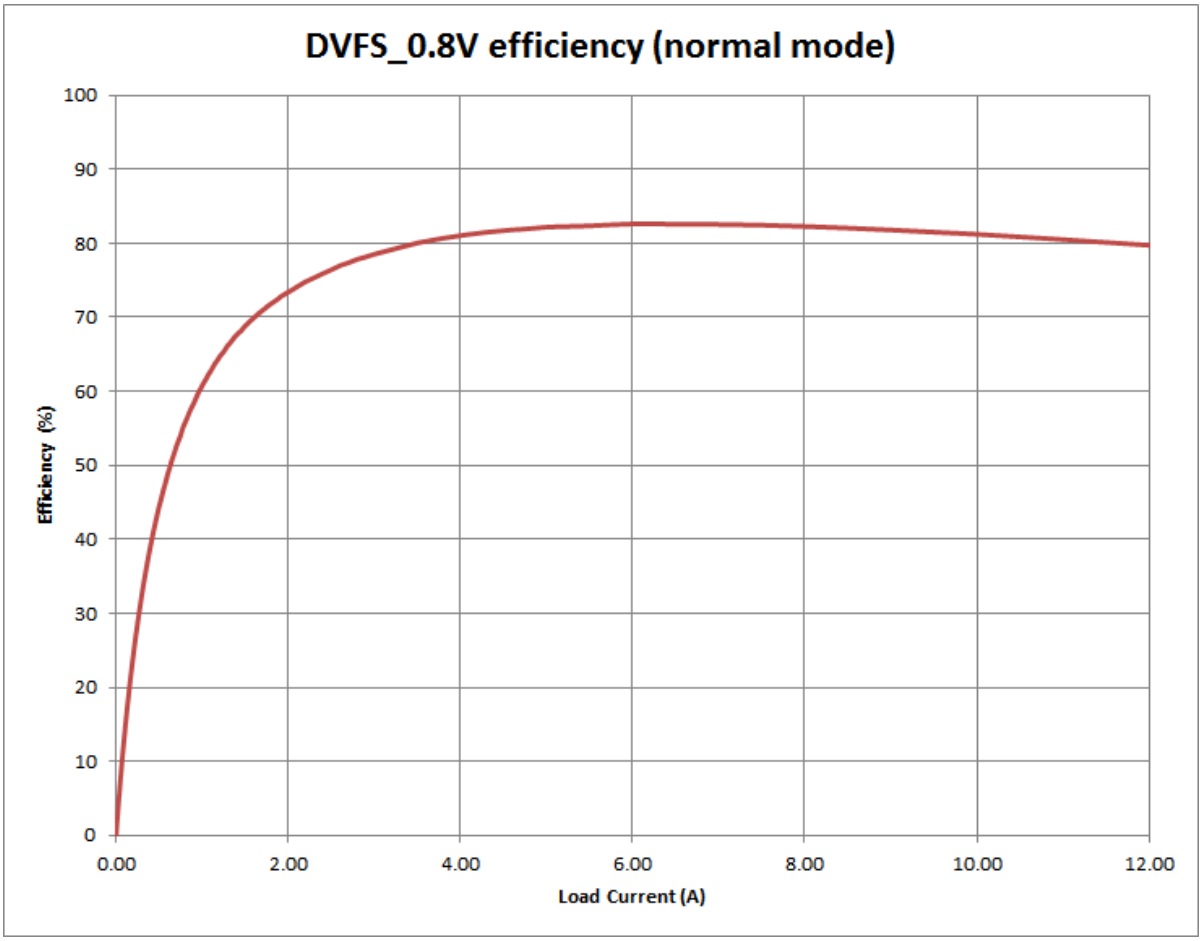

Figure 17: DVFS_08 Efficiency

Appendix A DA9063-72HO2-A Detailed Register Description

Key Settings

- Normal start-up
- Voltage monitor
- Buck Mem & Buck IO merged mode
- 2-wire control interface, standard speed
- RTC enabled
- LDO 3, 4, 6, 7, 8, and 9 GPIO controlled by host for 1.8 V or 3.3 V SD card supply select

Table 1: DA9063-72HO2-A Register Settings

Register Address	Function	Register Value	Register Description
0x00A	IRQ_MASK_A	0x00	nONKEY, RTC, and some status IRQ masks
0x00B	IRQ_MASK_B	0x10	Charger wakeup and temperature, current, or voltage IRQ masks
0x00C	IRQ_MASK_C	0x00	GPI7 to 0 and ADCIN1-3 IRQ masks
0x00D	IRQ_MASK_D	0x00	GPI15 to 8 and external control signal IRQ masks
0x00E	CONTROL_A	0x03	PSM target status, companion charger control
0x00F	CONTROL_B	0x09	Power-down / -up signalling
0x010	CONTROL_C	0x5B	Debounce, boot, DVC, and DEF_SUPPLY control
0x011	CONTROL_D	0x68	Watchdog and LED blink control
0x012	CONTROL_E	0x04	RTC, ecomode, feedback pins, V_LOCK
0x013	CONTROL_F	0x00	Watchdog reset, shutdown, and wakeup
0x014	PD_DIS	0x40	Disable / pause blocks when below the PSS sequencer PD_DIS slot
0x015	GPIO_0_1	0xDC	GPIO0 and 1 control
0x016	GPIO_2_3	0xED	GPIO2 and 3 control
0x017	GPIO_4_5	0x9E	GPIO4 and 5 control
0x018	GPIO_6_7	0xEF	GPIO6 and 7 control
0x019	GPIO_8_9	0xE4	GPIO8 and 9 control
0x01A	GPIO_10_11	0xE6	GPIO10 and 11 control
0x01B	GPIO_12_13	0xDF	GPIO12 and 13 control
0x01C	GPIO_14_15	0xFF	GPIO14 and 15 control registers
0x01D	GPIO_MODE0_7	0x80	GPIO0 to 7 mode control
0x01E	GPIO_MODE8_15	0x0E	GPIO8 to 15 mode control
0x01F	SWITCH_CONT	0xB0	Rail switches
0x020	BCORE2_CONT	0x00	BUCKCORE2 control
0x021	BCORE1_CONT	0x00	BUCKCORE1 control
0x022	BPRO_CONT	0x00	BUCKPRO control
0x023	BMEM_CONT	0x00	BUCKMEM control
0x024	BIO_CONT	0x00	BUCKIO control
0x025	BPERI_CONT	0x08	BUCKPERI control
0x026	LDO1_CONT	0x00	LDO1 control

Register Address	Function	Register Value	Register Description
0x027	LDO2_CONT	0x00	LDO2 control
0x028	LDO3_CONT	0x20	LDO3 control
0x029	LDO4_CONT	0x20	LDO4 control
0x02A	LDO5_CONT	0x80	LDO5 control
0x02B	LDO6_CONT	0x40	LDO6 control
0x02C	LDO7_CONT	0x40	LDO7 control
0x02D	LDO8_CONT	0x60	LDO8 control
0x02E	LDO9_CONT	0x60	LDO9 control
0x02F	LDO10_CONT	0x00	LDO10 control
0x030	LDO11_CONT	0x00	LDO11 control
0x031	SUPPLIES	0x00	Vibrator output level
0x032	DVC_1	0x00	Dynamic voltage control
0x033	DVC_2	0x00	Dynamic voltage control
0x034	ADC_MAN	0x20	ADC manual and automatic measurement control
0x035	ADC_CONT	0x01	ADC automatic measurement control
0x036	VSYS_MON	0xAA	
0x083	ID_2_1	0x00	PSS sequence control
0x084	ID_4_3	0xAA	PSS sequence control
0x085	ID_6_5	0xA1	PSS sequence control
0x086	ID_8_7	0xAA	PSS sequence control
0x087	ID_10_9	0xAA	PSS sequence control
0x088	ID_12_11	0x08	PSS sequence control
0x089	ID_14_13	0x08	PSS sequence control
0x08A	ID_16_15	0x87	PSS sequence control
0x08B	ID_18_17	0x48	PSS sequence control
0x08C	ID_20_19	0x49	PSS sequence control
0x08D	ID_22_21	0x00	PSS sequence control
0x08E	ID_24_23	0x06	PSS sequence control
0x08F	ID_26_25	0x00	PSS sequence control
0x090	ID_28_27	0x05	PSS sequence control
0x091	ID_30_29	0x09	PSS sequence control
0x092	ID_32_31	0x10	PSS sequence control
0x095	SEQ_A	0xDC	PSS sequencer slot end points
0x096	SEQ_B	0x4F	PSS sequencer slot end points
0x097	WAIT	0x10	Power sequencer wait cycle
0x098	EN_32K	0xEA	RTC clocking control
0x099	RESET	0x48	Reset timer control
0x09A	BUCK_ILIM_A	0xFF	Buck current limit

Register Address	Function	Register Value	Register Description
0x09B	BUCK_ILIM_B	0xFF	Buck current limit
0x09C	BUCK_ILIM_C	0xF5	Buck current limit
0x09D	BCORE2_CFG	0x81	BUCKCORE2 control
0x09E	BCORE1_CFG	0x81	BUCKCORE1 control
0x09F	BPRO_CFG	0x81	BUCKPRO control
0x0A0	BIO_CFG	0x81	BUCKPRO control
0x0A1	BMEM_CFG	0x81	BUCKMEM control
0x0A2	BPERI_CFG	0xA1	BUCKPERI control
0x0A3	VBCORE2_A	0x6C	BUCKCORE2 voltage A
0x0A4	VBCORE1_A	0x5A	BUCKCORE1 voltage A
0x0A5	VBPRO_A	0x7F	BUCKPRO voltage A
0x0A6	VBMEM_A	0x7D	BUCKMEM voltage A
0x0A7	VBIO_A	0x7D	BUCKIO voltage A
0x0A8	VBPERI_A	0x32	BUCKPERI voltage A
0x0A9	VLDO1_A	0x3C	LDO1 voltage A
0x0AA	VLDO2_A	0x3C	LDO2 voltage A
0x0AB	VLDO3_A	0x78	LDO3 voltage A
0x0AC	VLDO4_A	0x78	LDO4 voltage A
0x0AD	VLDO5_A	0x14	LDO5 voltage A
0x0AE	VLDO6_A	0x32	LDO6 voltage A
0x0AF	VLDO7_A	0x32	LDO7 voltage A
0x0B0	VLDO8_A	0x32	LDO8 voltage A
0x0B1	VLDO9_A	0x32	LDO9 voltage A
0x0B2	VLDO10_A	0x14	LDO10 voltage A
0x0B3	VLDO11_A	0x22	LDO11 voltage A
0x0B4	VBCORE2_B	0x3C	BUCKCORE2 voltage B
0x0B5	VBCORE1_B	0x5A	BUCKCORE1 voltage B
0x0B6	VBPRO_B	0x7F	BUCKPRO voltage B
0x0B7	VBMEM_B	0x7D	BUCKMEM voltage B
0x0B8	VBIO_B	0x7D	BUCKIO voltage B
0x0B9	VBPERI_B	0xB2	BUCKPERI voltage B
0x0BA	VLDO1_B	0x3C	LDO1 voltage B
0x0BB	VLDO2_B	0x3C	LDO2 voltage B
0x0BC	VLDO3_B	0x2D	LDO3 voltage B
0x0BD	VLDO4_B	0x2D	LDO4 voltage B
0x0BE	VLDO5_B	0x14	LDO5 voltage B
0x0BF	VLDO6_B	0x14	LDO6 voltage B
0x0C0	VLDO7_B	0x14	LDO7 voltage B

Register Address	Function	Register Value	Register Description
0x0C1	VLDO8_B	0x14	LDO8 voltage B
0x0C2	VLDO9_B	0x14	LDO9 voltage B
0x0C3	VLDO10_B	0x32	LDO10 voltage B
0x0C4	VLDO11_B	0x22	LDO11 voltage B
0x0C5	BBAT_CONT	0x00	Backup battery charger
0x0C6	GPO11_LED	0x00	High power GPO PWM
0x0C7	GPO14_LED	0x00	High power GPO PWM
0x0C8	GPO15_LED	0x00	High power GPO PWM
0x0C9	ADC_CFG	0xE0	ADC automatic measurement control
0x0CA	AUTO1_HIGH	0x00	ADC measurement thresholds
0x0CB	AUTO1_LOW	0x00	ADC measurement thresholds
0x0CC	AUTO2_HIGH	0x00	ADC measurement thresholds
0x0CD	AUTO2_LOW	0x00	ADC measurement thresholds
0x0CE	AUTO3_HIGH	0x00	ADC measurement thresholds
0x0CF	AUTO3_LOW	0x00	ADC measurement thresholds
0x105	INTERFACE	0xB9	Host interfaces
0x106	CONFIG_A	0x86	Host interfaces and other IOs
0x107	CONFIG_B	0x7F	VDD_FAULT comparator
0x108	CONFIG_C	0x50	Buck duty cycle and clock polarity
0x109	CONFIG_D	0x01	
0x10A	CONFIG_E	0xFF	BUCK and rail switch default settings
0x10B	CONFIG_F	0x07	LDO default and bypass mode settings
0x10C	CONFIG_G	0xFF	LDO default settings
0x10D	CONFIG_H	0xF0	
0x10E	CONFIG_I	0x04	
0x10F	CONFIG_J	0xE3	
0x110	CONFIG_K	0x80	GPIO pull resistors
0x111	CONFIG_L	0x0A	GPIO pull resistors
0x112	CONFIG_M	0x00	
0x113	CONFIG_N	0x00	
0x114	MON_REG_1	0x8A	
0x115	MON_REG_2	0xFC	
0x116	MON_REG_3	0x07	
0x117	MON_REG_4	0xF4	
0x121	GP_ID_0	0x01	
0x122	GP_ID_1	0x00	
0x123	GP_ID_2	0x00	
0x124	GP_ID_3	0x00	

Register Address	Function	Register Value	Register Description
0x125	GP_ID_4	0x00	
0x126	GP_ID_5	0x00	
0x127	GP_ID_6	0x00	
0x128	GP_ID_7	0x00	
0x129	GP_ID_8	0x00	
0x12A	GP_ID_9	0x00	
0x12B	GP_ID_10	0x00	
0x12C	GP_ID_11	0x00	
0x12D	GP_ID_12	0x00	
0x12E	GP_ID_13	0x00	
0x12F	GP_ID_14	0x00	
0x130	GP_ID_15	0x00	
0x131	GP_ID_16	0x00	
0x132	GP_ID_17	0x00	
0x133	GP_ID_18	0x00	
0x134	GP_ID_19	0x00	
0x183	CUSTOMER_ID	0x00	Chip ID
0x184	CONFIG_ID	0x72	Customer ID

Appendix B Software Implementation

B.1 Set DA9063 Register 0x94

For setting PMIC register 0x94, perform a write operation on register 0x94 and overwrite this register with a new value of 0x99. A typical pseudocode call would be:

```
int write(unsigned int reg, unsigned int val);

int err = 0;
if (err = write(0x94, 0x99)) {
       error("Unable to write register 0x94\n");
       return err;
}
```

B.2 Set DA9063 BKUP_TRG Bit or DA9063 GP_ID_1 Register

Depending upon which method is used (BKUP_TRG or GP_ID_1).

For setting BKUP_TRG, reference to the DA9063 Datasheet describes the register 0x1D (GPIO_MODE0_7). Bit 3 in this register is GPIO3_MODE and has the following settings:

- 0: GPI: debouncing off GPO: Sets output to low level (active low for sequencer control)
- 1: GPI: debouncing on GPO: Sets output to high level (active high for sequencer control)

Performing a read-modify-write operation on GPIO_MODE0_7 and enabling this GPIO3_MODE bit as 1 will set BKUP_TRG=1. A typical pseudocode call would be:

```
int update_bits(unsigned int reg, unsigned int mask, unsigned int val);

int err = 0;
if (err = update_bits(0x1D, 0x08, 0x08)) {
       error("Unable to update bit3 in register 0x1D\n");
       return err;
}
```

For setting GP_ID_1, reference to the DA9063 Datasheet describes the register 0x122 (GP_ID_1). Data from fuse array (OTP). Overwriting this register with a value of 0x01 will set GP_ID_1=1. Typical pseudocode would be:

```
int write(unsigned int reg, unsigned int val);

if (err = write(0x122, 0x01)) {
       error("Unable to write register 0x122\n");
       return err;
}
```

B.3 Set or Clear DA9063 BKUP_CTRL Bit

For setting BKUP_CTRL, reference to the DA9063 Datasheet describes the register 0x1D (GPIO_MODE0_7). Bit 4 in this register is GPIO4_MODE and has the following settings:

- 0: GPI: debouncing off GPO: Sets output to low level (active low for sequencer control)
- 1: GPI: debouncing on GPO: Sets output to high level (active high for sequencer control)

Performing a read-modify-write operation on GPIO_MODE0_7 and enabling this GPIO4_MODE bit as 1 will set BKUP_CTRL=1. A typical pseudocode call would be:

```
int update_bits(unsigned int reg, unsigned int mask, unsigned int val);

int err = 0;
if (err = update_bits(0x1D, 0x10, 0x10)) {
       error("Unable to update bit4 in register 0x1D\n");
       return err;
}
```

Performing a read-modify-write operation on GPIO_MODE0_7 and clearing this GPIO4_MODE bit as 0 will set BKUP_CTRL=0. A typical pseudocode call would be:

```
int update_bits(unsigned int reg, unsigned int mask, unsigned int val);

int err = 0;
if (err = update_bits(0x1D, 0x10, 0x00)) {
       error("Unable to update bit4 in register 0x1D\n");
       return err;
}
```

B.4 Read DA9063 BKUP_TRG Bit or DA9063 GP_ID_1 Register

Depending upon which method is used (BKUP_TRG or GP_ID_1).

For reading BKUP_TRG, reference to the DA9063 Datasheet describes the register 0x1D (GPIO_MODE0_7). Bit 3 in this register is GPIO3_MODE. Performing a read operation on GPIO_MODE0_7 and testing bit 3 will define the setting of BKUP_TRG. A typical pseudocode call would be:

```
int read(unsigned int reg, unsigned int *val);

int val = 0x00;

if (err = read(0x1D, &val)) {
       error("Unable to read register 0x1D\n");
       return err;
}
else {
       if (val & 0x08)
              print("WARM boot");
       else
              print("COLD boot\n");
}
```

Alternatively, if the SoC GPIO, GP1-08, is connected to BKUP_TRG output pin on DA9063 the SoC can directly determine the BKUP_TRG status by reading GP1-08.

For reading GP_ID_1, reference to the DA9063 Datasheet describes the register 0x122 (GP_ID_1). Data from fuse array (OTP). Reading this register with a non-zero value will define GP_ID_1=1. Typical pseudocode would be:

```
int read(unsigned int reg, unsigned int *val);

int val = 0x00;

if (err = read(0x122, &val)) {
       error("Unable to read register 0x122\n");
       return err;
}
else {
       if (val)
              print("WARM boot");
       else
              print("COLD boot\n");
}
```

B.5 Set DA9063 PMIC RTC Alarm

The operations for setting an RTC alarm in the DA9063 device are as follows.

Convert the contents of the general Linux RTC alarm structure held as values for seconds, minutes, hours, days, months and years into a simple data buffer which can be directly written to the alarm registers of the DA9063.

Reference to the DA9063 Datasheet provides a description of the RTC alarm registers 0x40 (COUNT_S) to 0x45 (COUNT_Y) inclusive – these registers form the length of the data block that should be written with the new alarm time. Typical pseudocode showing a conversion between the usual Linux RTC time structure and the writable DA9063 data buffer would be:

```
#define MONTHS_TO_DA9063(month)    ((month) + 1)
#define YEARS_TO_DA9063(year)      ((year) - 100)

struct rtc_time *tm;
/* seconds */
data[0] &= ~0x3F;
data[0] |= tm->tm_sec & 0x3F;
/* minutes */
data[1] &= ~0x3F;
data[1] |= tm->tm_min & 0x3F;
/* hours */
data[2] &= ~0x1F;
data[2] |= tm->tm_hour & 0x1F;
/* days */
data[3] &= ~0x1F;
data[3] |= tm->tm_mday & 0x1F;
/* months */
data[4] &= ~0x0F;
data[4] |= MONTHS_TO_DA9063(tm->tm_mon) & 0x0F;
/* years */
data[5] &= ~0x3F;
data[5] |= YEARS_TO_DA9063(tm->tm_year) & 0x3F;
```

Any pending alarms are stopped before the data buffer is written to the DA9063 RTC registers. Typical pseudocode would be:

```
int update_bits(unsigned int reg, unsigned int mask, unsigned int val);
/* stop alarm */
update_bits(0x4B, 0x40, 0x00);
```

A single bulk write function should be called. The data should be block written to the DA9063 device in a single I²C transfer. In this case the registers 0x40 (COUNT_S) to 0x45 (COUNT_Y) inclusive are written with the alarm data. Writing to the COUNT_Y register in the DA9063 will latch the all registers from COUNT_S and COUNT_Y into the current DA9063 RTC calendar counters. Typical pseudocode would be:

```
int bulk_write(unsigned int reg, const void *val, size_t val_count);
/* write alarm */
bulk_write(0x46, &data, 6);
```

B.6 Turn on DA9063 PMIC RTC Alarm

Reference to the DA9063 Datasheet describes the register 0x4B (ALARM_Y). Bit 6 in this register is described as ALARM_ON and has the following settings:

- 0: Alarm function is disabled
- 1: Alarm enabled

The Linux kernel device driver will set this bit using a read-modify-write operation to enable the RTC alarm in the DA9063. At the time of writing, the device driver uses the standard Linux regmap framework function for updating individual bits in a single register.

Typical pseudocode would be:

```
int update_bits(unsigned int reg, unsigned int mask, unsigned int val);
/* start alarm */
update_bits(0x4B, 0x40, 0x40);
```

B.7 Linux Device Driver PMIC RTC

Reference to the Linux kernel function for setting an RTC alarm in DA9063 is defined in the Opensource Linux device driver, inside the file `drivers/rtc/rtc-da9063.c`. The prototype for the DA9063 set alarm functions is given as `da9063_rtc_set_alarm()`. This DA9063 function has existed in the Opensource Linux kernel since Linux mainline v3.16-rc1. The Linux kernel mainline v4.16-rc1 has this function:

```
static int da9063_rtc_set_alarm(struct device *dev, struct rtc_wkalrm *alrm)
{
	struct da9063_compatible_rtc *rtc = dev_get_drvdata(dev);
	const struct da9063_compatible_rtc_regmap *config = rtc->config;
	u8 data[RTC_DATA_LEN];
	int ret;

	da9063_tm_to_data(&alrm->time, data, rtc);

	ret = da9063_rtc_stop_alarm(dev);
	if (ret < 0) {
		dev_err(dev, "Failed to stop alarm: %d\n", ret);
		return ret;
	}

	ret = regmap_bulk_write(rtc->regmap,
				config->rtc_alarm_secs_reg,
				&data[config->rtc_data_start],
				config->rtc_alarm_len);
	if (ret < 0) {
		dev_err(dev, "Failed to write alarm: %d\n", ret);
		return ret;
	}

	da9063_data_to_tm(data, &rtc->alarm_time, rtc);
```

```
	if (alrm->enabled) {
		ret = da9063_rtc_start_alarm(dev);
		if (ret < 0) {
			dev_err(dev, "Failed to start alarm: %d\n", ret);
			return ret;
		}
	}

	return ret;
}
```

The Linux kernel function for enabling the RTC alarm in the DA9063 is defined in the Opensource Linux device driver, inside the file `drivers/rtc/rtc-da9063.c`. The prototype for this function is given as `da9063_rtc_start_alarm()`. This DA9063 function has existed in the Opensource Linux kernel since Linux mainline v3.16-rc1. The Linux kernel mainline v4.16-rc1 has this function:

```
static int da9063_rtc_start_alarm(struct device *dev)
{
	struct da9063_compatible_rtc *rtc = dev_get_drvdata(dev);
	const struct da9063_compatible_rtc_regmap *config = rtc->config;

	return regmap_update_bits(rtc->regmap,
				  config->rtc_alarm_year_reg,
				  config->rtc_alarm_on_mask,
				  config->rtc_alarm_on_mask);
}
```

The Linux kernel function for reading the RTC time from the DA9063 is defined in the Opensource Linux device driver, inside the file `drivers/rtc/rtc-da9063.c`. The prototype for this function is given as `da9063_rtc_read_time()`. This DA9063 function has existed in the Opensource Linux kernel since Linux mainline v3.16-rc1. The Linux kernel mainline v4.16-rc1 has this function:

```
static int da9063_rtc_read_time(struct device *dev, struct rtc_time *tm)
{
	struct da9063_compatible_rtc *rtc = dev_get_drvdata(dev);
	const struct da9063_compatible_rtc_regmap *config = rtc->config;
	unsigned long tm_secs;
	unsigned long al_secs;
	u8 data[RTC_DATA_LEN];
	int ret;

	ret = regmap_bulk_read(rtc->regmap,
			       config->rtc_count_secs_reg,
			       data, RTC_DATA_LEN);
	if (ret < 0) {
		dev_err(dev, "Failed to read RTC time data: %d\n", ret);
		return ret;
	}

	if (!(data[RTC_SEC] & config->rtc_ready_to_read_mask)) {
		dev_dbg(dev, "RTC not yet ready to be read by the host\n");
		return -EINVAL;
	}

	da9063_data_to_tm(data, tm, rtc);

	rtc_tm_to_time(tm, &tm_secs);
	rtc_tm_to_time(&rtc->alarm_time, &al_secs);

	/* handle the rtc synchronisation delay */
```

```
	if (rtc->rtc_sync == true && al_secs - tm_secs == 1)
		memcpy(tm, &rtc->alarm_time, sizeof(struct rtc_time));
	else
		rtc->rtc_sync = false;

	return rtc_valid_tm(tm);
}
```

Revision History

Revision	Date	Description
1.0	28-Mar-2017	Initial version.
2.0	21-Mar-2018	Additional information regarding memory retention mode, reference designs and measurement results.
3.0	25-Feb-2022	File was rebranded with new logo, copyright and disclaimer

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance

Dialog Semiconductor's suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.

DA9063-A Power Management for R-Car H3 Platform

Important Notice and Disclaimer

(Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu

Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:

https://www.renesas.com/contact/