NAV GEN9 Single Dump Debug

API Specificiation

Rev. A

February 20, 2018

Submit technical questions at:

[callisto.hw@qualcomm.com](mailto:callisto.hw@qualcomm.com)

Revision History

|  |  |  |
| --- | --- | --- |
| Revision | Date | Description |
| A | 2/20/2018 | Initial draft, pre-release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table 1: Revision History

Contents

Contents

[1 Introduction 1-1](#_Toc508798635)

[1.1 Purpose 1-1](#_Toc508798636)

[1.2 Scope 1-1](#_Toc508798637)

[1.3 Terminology 1-2](#_Toc508798638)

[2 Features and Key Use Cases 2-3](#_Toc508798639)

[2.1 Standardized Application Programming Interface 2-3](#_Toc508798640)

[2.2 HW Debug Features 2-3](#_Toc508798641)

[3 Development Cycle 3-4](#_Toc508798642)

[3.1 HPG Development, Review and Sign-off 3-4](#_Toc508798643)

[3.2 DV Verification 3-4](#_Toc508798644)

[3.3 HVS Development and Verification on Vantage 3-5](#_Toc508798645)

[3.4 VI Development and Verification on RUMI and Silicon 3-5](#_Toc508798646)

[3.5 Translation of sequences into APIs 3-5](#_Toc508798647)

[3.6 Verification and Integration of APIs into SW 3-5](#_Toc508798648)

[3.7 Delivery of APIs to the SW team 3-6](#_Toc508798649)

[4 API Verification 4-6](#_Toc508798650)

[5 Application Programming Interface 5-6](#_Toc508798651)

[5.1 Dependencies 5-6](#_Toc508798652)

[5.2 Block Diagram 5-6](#_Toc508798653)

[5.3 Function Prototypes 5-7](#_Toc508798654)

[5.4 Sample Integration Code 5-12](#_Toc508798655)

Figures

[Figure 1: SDD API Development Cycle 3-4](#_Toc508798656)

[Figure 2: Nav SDD API Block Diagram 5-7](#_Toc508798657)

Tables

Flow Diagrams

# Introduction

## Purpose

This document is a specification for the definition, development and deployment of the Nav Single Dump Debug API. It describes in detail the scope of the work, how the API will be developed, recommendations on integration and the management of release to the software team.

## Scope

The content of this document is intended for internal distribution to engineering teams involved in the development, integration, and deployment of QCT ASICs. References

| Ref. | Document | | |
| --- | --- | --- | --- |
| QUALCOMM | | | |
| Q1 | Callisto Navigation Receiver: HDD |  |
| Q2 | Software Interface Documents |  |
| Q3 | Nav Gen9 HPG |  |
|  |  |  |
| Standards | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Resources | | |
|  |  |  |

Table 2: References

## Terminology

| Acronym or Term | Definition |
| --- | --- |
| API | Application Programming Interface |

Table 3: Acronym or Term

# 

# Features and Key Use Cases

The Single Dump Debug API is intended to be an abstraction layer between Gen9 SW and the underlying HW implementation of debug features. The scope of this API will be limited to the enablement of HW debug features and the collection of debug information from that HW. The primary goals for this API is to eliminate changes to SW in the case that a target HW has changed and create a standardized debug data set to facilitate SW/HW debug of complex issues. This will make any future HW changes transparent to the SW team in these key areas.

## Standardized Application Programming Interface

The Single Dump Debug API will implement a standardized application programming interface to provide abstraction of HW-specific details that may change from target to target. By providing this abstraction layer, the SW implementation of these key areas can be greatly simplified and streamlined.

## HW Debug Features

A set of functions will be provided to use all HW debug features, as defined in the Single Dump Debug process. These functions will provide methods to enable/disable HW debug features, extract debug information from the HW and parse the extracted data.

# 

# Development Cycle

The development cycle will be split into seven phases:

* HPG development, review and sign-off (HW/SW teams)
* DV verficiation of HPG sequences (DV team)
* HVS development and verficiation of HPG sequences on Vantage (HW/VI teams)
* VI verification of HPG sequences on RUMI and Silicon (VI team)
* Translation of sequences into APIs (HW/VI team)
* Delivery of APIs to SW (HW team)
* Verification and Integration of APIs into SW (SW team)

Each of these phases may be augmented with sub-phases as the process develops over time. The overall flow is as follows:



Figure 1: SDD API Development Cycle

## HPG Development, Review and Sign-off

The first step in the development cycle is the HPG development. This will occur at the beginning of a design cycle and is fully owned by the HW team. When the HPG is complete, a review will be held with the HW and SW teams. Upon completion of this review, the HPG will be signed-off which will trigger the next phase of development. Note that the HPG itself will have milestones (M0-M3) during HW development, and this phase may need to be revisited for some or all of those milestones.

## DV Verification

After HPG sign-off the DV team will implement the HPG sequences in System Verilog and run simulations to verify the sequences. Any issues found at this point will feed back to phase 1, then feed forward back to this phase. A Q\* milestone (TBD, Q1?) will indicate that this phase is complete and all HPG sequences are fully verified.

## HVS Development and Verification on Vantage

After DV verification is complete, the HPG sequences will be translated from System Verilog into stand-alone C code that will be tested on the Vantage FPGA platform. In some cases, such as PLL configuration, HW features will not be available for implementation and testing and thus, this step will be a partial implementation of the sequences. Additionally, Vantage does not support all silicon targets, in which case this step will be skipped altogether.

## VI Development and Verification on RUMI and Silicon

The next phase will be to either augment the HVS C code, or translate System Verilog from DV into VI for verification on RUMI, depending on the target and Vantage FPGA support. An important note for this step is that the C code written here is to be considered completely stand-alone, outside of address and register macro definitions provided by the HW team.

This phase will be split between pre-Si activities on RUMI and SoD activities on silicon. The key here is that the C code and testing will remain the same between pre- and post-Si.

## Translation of sequences into APIs

At this point, the sequences have been verified on several platforms and are considered ready for translation into APIs. This will consist of working between the Nav SW team to define the interfaces, and with CoreBSP/DAL API teams to implement production quality code. In this phase, the APIs will have many external dependencies and should be expected to adhere 100% to any coding guidelines and coding styles required.

It is expected that this phase will be completed on RUMI (full-stack) and should be ready before SoD.

Details TBD.

## Verification and Integration of APIs into SW

In this phase, the APIs will be verified through extensive unit testing (TBD?). Upon completion of unit testing, they will be integrated into the main SW codebase and again verified through extensive testing within the Nav SW test team (TBD). One point to consider here is that the impact to test teams should be minimized to the point that we will need sign-off on this verification, but no additional testing efforts should be spent by the test team.

Details TBD.

## Delivery of APIs to the SW team

The final phase will be the official delivery of APIs to the SW team. This step should be mostly a formality, and should include a review of the APIs, any changes from previous versions, and inclusion of sign-offs for each of the previous phases.

# API Verification

The API will be verified across several platforms before delivery to the SW team. The process will start with the HPG, fully reviewed and signed off. The seqences will then be verified in DV, HVS on Vantage, VI on RUMI and VI on each target. Any changes will be fed back to the HPG, then fed forward through DV, HVS and VI.

Details TBD.

# Application Programming Interface

## Dependencies

The SDD API will require the following dependencies:

* nav\_hwio/msmhwio/HALhwio for reading and writing Nav DBE registers
* gps\_types.h for data type definitions

## Block Diagram

The SDD API is implemented as a 2-tier module. The top level, navsdd\_api contains all the exposed functionality to the calling module. The lower level, navdd contains the internal functions of the API that are not exposed to external modules.



Figure 2: Nav SDD API Block Diagram

## Function Prototypes

/\*\*

\* This function initializes the SDD API

\*

\* @return boolean Success or failure

\*/

boolean navsdd\_Init(void);

/\*\*

\* This function enables the Nav SDD HW by traversing the HW

\* hierarchy and enabling HW modules that are powered on, clocked and not in reset

\* @param q\_HwConfig - HW configuration ID. 0 is default user mode, other

\* values are TBD or will be used for debug

\*

\* @return boolean Success or failure

\*/

boolean navsdd\_HwEnable(uint32 q\_HwConfig);

/\*\*

\* This function disables the Nav SDD HW by traversing the HW

\* hierarchy and disabling HW modules that are powered on, clocked and not in reset

\*

\* @return boolean Success or failure

\*/

boolean navsdd\_HwDisable(void);

/\*\*

\* This function reads the Nav SDD HW information and stores it in the buffer

\* pointed to by p\_CollectionData

\* @param p\_CollectionData - pointer to collection buffer allocated by the

\* caller, of size C\_NAVSDD\_COLLECTION\_UINTS

\* @param q\_BufferSize - size of the collection buffer, in bytes

\*

\* @return uint32 The number of valid uint32 words collected

\*/

uint32 navsdd\_HwCollect(uint32 \*p\_CollectionData, uint32 q\_BufferSize);

/\*\*

\* This function reads the Nav SDD subsystem information and stores it in the buffer

\* pointed to by p\_CollectionData

\* @param p\_CollectionData - pointer to collection buffer allocated by the

\* caller, of size C\_NAVSDD\_COLLECTION\_UINTS

\* @param q\_BufferSize - size of the collection buffer, in bytes

\* @param e\_SubsystemId - Subsystem ID. See navsdd\_SubsystemIdType

\*

\* @return uint32 The number of valid uint32 words collected

\*/

uint32 navsdd\_SubsystemHwCollect(uint32 \*p\_CollectionData, uint32 q\_BufferSize, navsdd\_SubsystemIdType e\_SubsystemId);

/\*\*

\* This function returns the number of uint32 words collected by the subsystem

\* for a given subsystem ID

\* @param q\_HwConfig - HW configuration ID. 0 is default user mode, other

\* values are TBD or will be used for debug

\* @param e\_SubsystemId - Subsystem ID. See navsdd\_SubsystemIdType

\*

\* @return uint32 The number of uint32 words collected for the given HwConfig and Subsystem

\*/

uint32 navsdd\_GetNumSubsystemCollectWords(uint32 q\_HwConfig, navsdd\_SubsystemIdType e\_SubsystemId);

/\*\*

\* This function reads a portion of Nav internal memory and stores it in the buffer

\* pointed to by p\_CollectionData

\* @param p\_CollectionData - pointer to collection buffer allocated by the

\* caller, of size C\_NAVSDD\_COLLECTION\_UINTS

\* @param q\_BufferSize - size of the collection buffer, in bytes

\* @param p\_StartAddress - pointer to the starting read address

\*

\* @param q\_NumWords - number of 32-bit words to read from the memory location

\*

\* @return uint32 The number of valid uint32 words collected

\*/

uint32 navsdd\_NavMemCollect(uint32 \*p\_CollectionData, uint32 q\_BufferSize, uint32 \*p\_StartAddress, uint32 q\_NumWords);

## Sample Integration Code

#include "navSdd.h"

uint32 q\_sddCollectionData[C\_NAVSDD\_COLLECTION\_UINTS] = {0};

uint32\* p\_sddCollectionData = q\_sddCollectionData;

uint32 u\_HwConfig = 0;

uint32 u\_sddLoop = 0;

// Initialize the API - one time call at startup

navSdd\_Init();

// Enable the SDD HW - one time call at HW power-on

navSdd\_HwEnable(0);

uint32 q\_sddCollectionData[C\_NAVSDD\_COLLECTION\_UINTS] = {0};

uint32\* p\_sddCollectionData = q\_sddCollectionData;

uint32 u\_HwConfig = 0; // TODO: Add global that tracks the current HwConfig, pass into collect function

uint32 u\_sddLoop = 0;

uint32 q\_NumWords = 0;

if(navsdd\_HwCollect(p\_sddCollectionData, sizeof(q\_sddCollectionData)))

{

// This for loop overwhelms the UART on the lean build, we can only get ~44 messages out before dropping

for(u\_sddLoop = 0; u\_sddLoop < 10; u\_sddLoop++)

{

CC\_MSG\_HIGH("GPS\_DMOV: Nav SDD collect word %d: 0x%x", u\_sddLoop, q\_sddCollectionData[u\_sddLoop],0);

}

} else

{

CC\_MSG\_HIGH("GPS\_DMOV: Nav SDD HW Collect failed!!!",0,0,0);

}

q\_NumWords = navsdd\_GetNumSubsystemCollectWords(0, NAVSDD\_CP);

CC\_MSG\_HIGH("GPS\_DMOV: SDD: Num CP Words0x%x", q\_NumWords, 0,0);

if(navsdd\_SubsystemHwCollect(p\_sddCollectionData, sizeof(q\_sddCollectionData), NAVSDD\_CP))

{

// This for loop overwhelms the UART on the lean build, we can only get ~44 messages out before dropping

for(u\_sddLoop = 0; u\_sddLoop < 10; u\_sddLoop++)

{

CC\_MSG\_HIGH("GPS\_DMOV: Nav SDD Subsystem collect word %d: 0x%x", u\_sddLoop, q\_sddCollectionData[u\_sddLoop],0);

}

} else

{

CC\_MSG\_HIGH("GPS\_DMOV: Nav SDD Subsystem HW Collect failed!!!",0,0,0);

}

if(navsdd\_NavMemCollect(p\_sddCollectionData, sizeof(q\_sddCollectionData), (uint32\*) 0x50140000, 10))

{

// This for loop overwhelms the UART on the lean build, we can only get ~44 messages out before dropping

for(u\_sddLoop = 0; u\_sddLoop < 10; u\_sddLoop++)

{

CC\_MSG\_HIGH("GPS\_DMOV: Nav SDD SM collect word %d: 0x%x", u\_sddLoop, q\_sddCollectionData[u\_sddLoop],0);

}

} else

{

CC\_MSG\_HIGH("GPS\_DMOV: Nav SDD SM HW Collect failed!!!",0,0,0);

}

// Change the SDD HW config - behavior TBD, value given as an example

navSdd\_HwEnable(0x1F3C)

// Disable the SDD HW if desired

navSdd\_HwDisable(0);

List of Collection Registers and HW Configs

The following is a list of the registers collected by each subsystem and each HW config. Initially, only HW config 0 is supported. This list will expand as additional HW configs are added.

The lists are stored in the API as 2-D arrays of addresses.

uint32 q\_versionId[1][1] =

{

{

C\_NAVSDD\_HW\_VERSION\_MAJOR << 24 | C\_NAVSDD\_HW\_VERSION\_MINOR << 16 | C\_NAVSDD\_API\_REVISION

}

};

uint32 q\_clockCollectRegs[1][C\_NAVSDD\_I\_CLOCK\_COLLECTION\_UINTS] =

{

// Config 0

{

HWIO\_ADDR(NAV\_CC\_NAV\_SS\_BCR),

HWIO\_ADDR(NAV\_CC\_NAV\_SS\_GDSCR),

HWIO\_ADDR(NAV\_CC\_NAV\_SS\_CFG\_GDSCR),

HWIO\_ADDR(NAV\_CC\_NAV\_SS\_CFG2\_GDSCR),

HWIO\_ADDR(NAV\_CC\_BB\_MAIN\_CMD\_RCGR),

HWIO\_ADDR(NAV\_CC\_BB\_MAIN\_CFG\_RCGR),

HWIO\_ADDR(NAV\_CC\_BB\_CORE\_CBCR),

HWIO\_ADDR(NAV\_CC\_BB\_CORE\_SREGR),

HWIO\_ADDR(NAV\_CC\_CP\_CBCR),

HWIO\_ADDR(NAV\_CC\_DMA\_CBCR),

HWIO\_ADDR(NAV\_CC\_DP\_CBCR),

HWIO\_ADDR(NAV\_CC\_BB\_QLINK\_CBCR),

HWIO\_ADDR(NAV\_CC\_BB\_QLINK\_SREGR),

HWIO\_ADDR(NAV\_CC\_WB\_CBCR),

HWIO\_ADDR(NAV\_CC\_MDM\_CBCR),

HWIO\_ADDR(NAV\_CC\_SNOC\_CBCR),

HWIO\_ADDR(NAV\_CC\_STMR\_XO\_CBCR),

HWIO\_ADDR(NAV\_CC\_XO\_GDSC\_CBCR),

HWIO\_ADDR(NAV\_CC\_NAV\_IPCAT),

HWIO\_ADDR(NAV\_CC\_DEBUG\_MUX\_MUXR),

HWIO\_ADDR(NAV\_CC\_PLL\_TEST\_MUX\_MUXR),

HWIO\_ADDR(NAV\_CC\_PLL\_RESET\_N\_MUXR),

HWIO\_ADDR(NAV\_CC\_PLL\_STATUS\_MUXR),

HWIO\_ADDR(NAV\_CC\_SPARE\_CTRL),

HWIO\_ADDR(NAV\_CC\_SPARE\_STATUS),

HWIO\_ADDR(NAV\_CC\_NAV\_AXI\_BRIDGE\_STATUS),

HWIO\_ADDR(NAV\_CC\_NAV\_CLK\_STATUS),

HWIO\_ADDR(NAV\_CC\_SNOC\_QSB\_SIDEBAND),

HWIO\_ADDR(NAV\_CC\_CLK\_MISC),

HWIO\_ADDR(NAV\_CC\_TEST\_DBG\_CONFIG),

HWIO\_ADDR(NAV\_CC\_QDSS\_TESTBUS),

HWIO\_ADDR(NAV\_CC\_AXI2SMI\_STATUS),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_MODE),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_L\_VAL),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_CAL\_L\_VAL),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_USER\_CTL),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_USER\_CTL\_U),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_USER\_CTL\_U1),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_CONFIG\_CTL),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_CONFIG\_CTL\_U),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_CONFIG\_CTL\_U1),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_TEST\_CTL),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_TEST\_CTL\_U),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_TEST\_CTL\_U1),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_STATUS),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_FREQ\_CTL),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_OPMODE),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_STATE),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_ALPHA\_VAL),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_SPARE),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_SSC\_DELTA\_ALPHA),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_SSC\_UPDATE\_RATE),

HWIO\_ADDR(NAV\_CC\_PLL4\_PLL\_SSC\_NUM\_STEPS)

}

// Future HW configs here

};

uint32 q\_qlinkCollectRegs[1][C\_NAVSDD\_I\_QLINK\_COLLECTION\_UINTS] =

{

// Config 0

{

HWIO\_ADDR(NQ\_CONFIG),

HWIO\_ADDR(NQ\_FIFO\_CTRL),

HWIO\_ADDR(NQ\_FIFO\_STATUS),

HWIO\_ADDR(NQ\_DEBUG\_SHADOW\_REG)

}

// Future HW configs here

};

uint32 q\_dmaCollectRegs[1][C\_NAVSDD\_I\_DMA\_COLLECTION\_UINTS] =

{

// Config 0

{

HWIO\_ADDR(DMACTRL\_GLB\_STATUS),

HWIO\_ADDR(DMACTRL\_GLB\_CFG),

HWIO\_ADDR(DMACTRL\_EXT\_MEM\_BASE\_ADDR),

HWIO\_ADDR(DMACTRL\_INT\_MEM\_BASE\_ADDR),

HWIO\_ADDR(DMACTRL\_LAST\_ADDR\_M0),

HWIO\_ADDR(DMACTRL\_LAST\_CTRL\_M0),

HWIO\_ADDR(DMACTRL\_LAST\_ADDR\_M1),

HWIO\_ADDR(DMACTRL\_LAST\_CTRL\_M1),

HWIO\_ADDR(DMACTRL\_LAST\_ADDR\_M2),

HWIO\_ADDR(DMACTRL\_LAST\_CTRL\_M2),

HWIO\_ADDR(DMACTRL\_LAST\_ADDR\_M3),

HWIO\_ADDR(DMACTRL\_LAST\_CTRL\_M3),

HWIO\_ADDR(BUS\_STATS\_STATUS),

HWIO\_ADDR(DMACTRL\_BUP\_STATUS1),

HWIO\_ADDR(DMACTRL\_BUP\_STATUS2),

HWIO\_ADDR(DMACTRL\_BUP\_STATUS3),

HWIO\_ADDR(DMACTRL\_BUP\_STATUS4),

HWIO\_ADDR(DMACTRL\_BUP\_STATUS5),

HWIO\_ADDR(DMACTRL\_BUP\_STATUS6),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD1\_START\_ADDR, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD2\_START\_ADDR, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD3\_START\_ADDR, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD\_READY, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_CTL, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_CFG, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS1, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS2, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_SRC\_ADDR, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_DST\_ADDR, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W0, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W1, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W2, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_MIN, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_MAX, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_AVG, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_CUR, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS3, 0),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD1\_START\_ADDR, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD2\_START\_ADDR, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD3\_START\_ADDR, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD\_READY, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_CTL, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_CFG, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS1, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS2, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_SRC\_ADDR, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_DST\_ADDR, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W0, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W1, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W2, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_MIN, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_MAX, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_AVG, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_CUR, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS3, 1),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD1\_START\_ADDR, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD2\_START\_ADDR, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD3\_START\_ADDR, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD\_READY, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_CTL, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_CFG, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS1, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS2, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_SRC\_ADDR, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_DST\_ADDR, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W0, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W1, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W2, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_MIN, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_MAX, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_AVG, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_CUR, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS3, 2),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD1\_START\_ADDR, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD2\_START\_ADDR, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD3\_START\_ADDR, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_CMD\_READY, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_CTL, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_CFG, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS1, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS2, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_SRC\_ADDR, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_DST\_ADDR, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W0, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W1, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_LAST\_CC\_W2, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_MIN, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_MAX, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_AVG, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_STATS\_CUR, 3),

HWIO\_ADDRI(DMACTRL\_CHn\_STATUS3, 3)

// Cache data/buffer data will be dumped separately

}

// Future HW configs here

};

uint32 q\_cpCollectRegs[1][C\_NAVSDD\_I\_CP\_COLLECTION\_UINTS] =

{

// Config 0

{

HWIO\_ADDR(CCP\_IND\_WORDS\_NEEDED),

HWIO\_ADDR(CCP\_DIR\_WORDS\_NEEDED),

HWIO\_ADDR(CCP\_IND\_MOVE\_SIZE\_RMM),

HWIO\_ADDR(CCP\_DIR\_MOVE\_SIZE\_RMM),

HWIO\_ADDR(CCP\_CP\_CLKEN),

HWIO\_ADDR(CCP\_PROG\_STATUS),

HWIO\_ADDR(CCP\_G10\_STATUS),

HWIO\_ADDR(CCP\_G32\_STATUS),

HWIO\_ADDR(CCP\_G54\_STATUS),

HWIO\_ADDR(CCP\_G76\_STATUS),

HWIO\_ADDR(CCP\_G98\_STATUS),

HWIO\_ADDR(CCP\_GBA\_STATUS),

HWIO\_ADDR(CCP\_GDC\_STATUS),

HWIO\_ADDR(CCP\_P10\_STATUS),

HWIO\_ADDR(CCP\_P2\_STATUS),

HWIO\_ADDR(CCP\_GNSS\_RTC\_STATUS),

HWIO\_ADDR(CCP\_RMM\_IND\_1\_STATUS),

HWIO\_ADDR(CCP\_RMM\_IND\_2\_STATUS),

HWIO\_ADDR(CCP\_RMM\_DIR\_1\_STATUS),

HWIO\_ADDR(CCP\_RMM\_DIR\_2\_STATUS),

HWIO\_ADDR(CCP\_STMR\_RTC\_STATUS),

HWIO\_ADDR(CCP\_IRQ\_GEN),

HWIO\_ADDR(CCP\_GPIO\_TO\_DCP),

HWIO\_ADDR(CCP\_GPIO\_FROM\_DCP\_STATUS),

HWIO\_ADDR(CCP\_RMM\_CONFLICT\_STATUS),

HWIO\_ADDR(CCP\_RMM\_POINTER\_INIT),

HWIO\_ADDR(CP\_BUF2\_BASE\_ADDR),

HWIO\_ADDR(CP\_STATUS)

}

// Future HW configs here

};

uint32 q\_dpCollectRegs[1][C\_NAVSDD\_I\_DP\_COLLECTION\_UINTS] =

{

// Config 0

{

HWIO\_ADDR(DP\_TC\_CTL),

HWIO\_ADDR(DP\_TC\_STATUS),

HWIO\_ADDR(DP\_MEM\_START\_ADDR),

HWIO\_ADDR(DP\_TMPM\_START\_ADDR),

HWIO\_ADDR(DP\_STATUS),

HWIO\_ADDR(DP\_GNSS\_RTC\_STATUS),

HWIO\_ADDR(DP\_STMR\_RTC\_STATUS),

HWIO\_ADDR(DP\_DCP\_PROG\_STATUS),

HWIO\_ADDR(DP\_DCP\_G10\_STATUS),

HWIO\_ADDR(DP\_DCP\_G32\_STATUS),

HWIO\_ADDR(DP\_DCP\_G54\_STATUS),

HWIO\_ADDR(DP\_DCP\_G76\_STATUS),

HWIO\_ADDR(DP\_DCP\_G98\_STATUS),

HWIO\_ADDR(DP\_DCP\_GBA\_STATUS),

HWIO\_ADDR(DP\_DCP\_GDC\_STATUS),

HWIO\_ADDR(DP\_DCP\_P10\_STATUS),

HWIO\_ADDR(DP\_DCP\_P2\_STATUS),

HWIO\_ADDR(DP\_DCP\_IRQ\_GEN),

HWIO\_ADDR(DP\_DCP\_GPIO\_TO\_CCP),

HWIO\_ADDR(DP\_DCP\_GPIO\_FROM\_CCP\_STATUS)

}

// Future HW configs here

};

uint32 q\_miscCollectRegs[1][C\_NAVSDD\_I\_MISC\_COLLECTION\_UINTS] =

{

// Config 0

{

HWIO\_ADDR(RC\_CONTROL),

HWIO\_ADDR(RC\_TC\_TXFR\_CTL),

HWIO\_ADDR(RC\_TC\_TXFR\_CFG0),

HWIO\_ADDR(RC\_TC\_TXFR\_CFG1),

HWIO\_ADDR(RC\_TC\_TXFR\_STATUS),

HWIO\_ADDR(RC\_GNSS\_RTC\_STATUS),

HWIO\_ADDR(STMR\_STATUS3),

HWIO\_ADDR(RC\_IRQ\_STATUS1),

HWIO\_ADDR(RC\_IRQ\_STATUS2),

HWIO\_ADDR(SMI\_MODE),

HWIO\_ADDR(SMI\_SCH\_MODE),

HWIO\_ADDR(SMI\_BASEADDR1\_BANK0),

HWIO\_ADDR(SMI\_BASEADDR2\_BANK0),

HWIO\_ADDR(SMI\_BASEADDR3A\_BANK0),

HWIO\_ADDR(SMI\_BASEADDR3B\_BANK0),

HWIO\_ADDR(SMI\_BASEADDR3C\_BANK0),

HWIO\_ADDR(SMI\_BASEADDR3D\_CX20),

HWIO\_ADDR(SMI\_BASEADDR4),

HWIO\_ADDR(SMI\_BASEADDR1\_BANK1),

HWIO\_ADDR(SMI\_BASEADDR2\_BANK1),

HWIO\_ADDR(SMI\_BASEADDR3A\_BANK1),

HWIO\_ADDR(SMI\_BASEADDR3B\_BANK1),

HWIO\_ADDR(SMI\_BASEADDR3C\_BANK1),

HWIO\_ADDR(SMI\_BASEADDR2P),

HWIO\_ADDR(SMI\_RSTRZN\_CFG),

HWIO\_ADDR(SMI\_RSTRZN\_ADDR),

HWIO\_ADDR(SMI\_RSTRZN\_STATUS)

}

// Future HW configs here

};