The ICM-20648 and ICM-20948 DMP User Guide

INTRODUCTION

Ever since InvenSense introduce the world's first 3 axis gyro in the MPU-3000, the DMP (Digital Motion Processor) hardware has been a fixture in several of its motion MEMS devices. The concept of the DMP is to provide on-chip motion processing capabilities that will off-load the computation from the external processor. This will give more flexibility for the external processor to save system power and also gives customers additional hardware capabilities in the motion device such as sensor fusion and gesture detection.

Because the DMP is a very powerful hardware tool and a prominent feature in the TDK-InvenSense motion MEMS devices, there is a lot of questions from customers on how to use it and integrate it into their applications. This document is made to help answer some of those questions.

This document will focus on the ICM-20648 and the ICM-20948 which is the latest MEMs motion devices with the DMP integrated. The ICM-20648 and the ICM-20948 has the latest and most powerful DMP which also has the most complexity in terms of integration into customer applications. 90% of all DMP questions are related to these 2 products.

This document will attempt to capture the most asked questions of the DMP. It will go through the DMP flow from initialization, configuration, and output. It will give code examples from the TDK-InvenSense released code of the eMD-SmartMotion-ICM20948-1.1.0-MP.

TDK Corporation
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
www.invensense.com

Document Number: AN-000246
Revision: 1.0
Release Date: 10/08/2020

CONTENTS

DMP FAQ

This section covers the most common questions customers typically asks about the DMP.

WHAT EXACTLY IS THE DMP? IS IT A FULL PROCESSOR WITH PROCESSOR CAPABILITIES?

The DMP, or Digital Motion Processor, is a TDK-InvenSense proprietary developed hardware embedded into several of its inertial motion MEMS sensor. The main purpose of the DMP is to process on-chip the motion sensor data information to provide sensor fusion data information and gesture detection data for customer applications.

The DMP is definitely not a processor. There has limited memory, no non-volatile memory, no IDE for customer programmability, and no GPIOs. You can think of the DMP as a specialized FPGA for sensor data processing.

WHAT ARE THE DMP FEATURES? WHAT IS SENSOR FUSION? WHAT GESTURES CAN THE DMP DETECT?

Because the DMP evolves both hardware and software, different DMP has different features and also different performances. The DMP features when dealing with motion are usually the following 3 categories

- Sensor Dynamic Calibration : Algorithms which run in the background and dynamically calibrates the accel, gyro, or mag while the device is in use.
- Sensor Fusion : Where accel, gyro, or mag data is taken and mathematically integrated together to calculate more relevant data for applications. In this case the motion, common sensor fusion data are typically Quaternions, Euler Angles, or linear translations.
- Gesture Recognition : Where motion data is used to identify certain motions or activities. Examples include pedometer, pick up, run, walk, etc….

For the ICM-20648 and ICM-20948 SmartMotion eMD code release 1.x. The DMP features included are

ICM-20x48 DMP Features	Description
Pedometer	Step Count and Step Detection
Significant Motion Detection	Used for wake up functionality
Pick Up	Detects when device is picked up
Bring To See	Detects motion of bringing a watch-like device to see
Tilt	Detects when device tilts past a threshold
Basic Activity Classifiers	Detects Run, Walk, Bike, Stand, and Drive activity
6-Axis Sensor Fusion	Accel and Gyro quaternion generation
9-Axis Sensor Fusion	Accel and Gyro and Compass quaternion generation
Dynamic Accel Cal	A constant and in use calibration of accel. Reduces errors due to wear and tear
Dynamic Gyro Cal	A constant and in use calibration of gyro. Reduces errors due to temperature and reference drift
Dynamic Compass Cal	A constant and in use calibration of compass. Reduces errors due to magnetic variations

As you can see, the DMP features are very much related to Android device features for obvious reasons. Android devices are widely prevalent in the market and Android also provide testing and certification procedures for TDK-InvenSense to set as a passing goal.

HOW DO YOU KNOW A SENSOR IS CALIBRATED?

Base sensors (accel, gyro, mag) and GRV has an associated accuracy flag which allows users to know the calibration status. The accuracy flag is between 0 and 3. An accuracy of ‘3’ means the sensor is calibrated. An accuracy of ‘0’ means the sensor is not calibrated. Certain sensors will use the accuracy of ‘2’ to indicate data is calibrated but is not optimal.

For example, the calibrated mag –

'0' = Mag is not calibrated. Raw values are being outputted
'2' = Mag was calibrated however there is a magnetic disturbance detected which will result in errors.
'3' = Mag calibrated

The accuracy flag is calculated and outputted by the DMP. However not all sensors will have accuracy flags. Sensors like RV accuracy flags are always '0', instead they use a heading accuracy error which outputs and estimated error in radians.

CAN I DISABLE DMP DYNAMIC CALIBRATION OF THE SENSORS?

Yes. The dynamic calibration of accel, gyro, and compass is a DMP feature and can be enabled or disabled. By default the eMD will enable the calibration. However if you want to disable it you can look where this API is being access

```
/**
* Sets motion event control register.
* @param[in] output_mask   Turns features on/off according to following bit definition,
*                                                  bit set indicates on, bit clear indicates off.
*
*   BAC_WEAR_EN         0x8000 - change BAC behavior for wearable platform
*      PEDOMETER_EN          0x4000 - pedometer engine
*      PEDOMETER_INT_EN      0x2000 - pedometer step detector interrupt
*      SMD_EN                      0x0800 - significant motion detection interrupt
*      ACCEL_CAL_EN          0x0200 - accel calibration
*      GYRO_CAL_EN                 0x0100 - gyro calibration
*      COMPASS_CAL_EN              0x0080 - compass calibration
*      NINE_AXIS_EN        0x0040 - 9-axis algorithm execution
*      GEOMAG_EN                   0x0008 - Geomag algorithm execution
*      BTS_LTS_EN          0x0004 - bring & look to see
*      BAC_ACCEL_ONLY_EN   0x0002 - run BAC as accel only
*/
int dmp_icm20948_set_motion_event_control(struct inv_icm20948 * s, unsigned short output_mask)
```

Just make sure when this API is called (during sensor enabled) that you do not enable ACCEL_CAL_EN or GYRO_CAL_EN or COMPASS_CAL_EN.

HOW DOES DMP OUTPUT DATA? - INTERRUPTS, FIFO, AND PEDOMETER

The DMP output data in 3 methods.

Interrupts – the DMP uses INT1 pin of the ICM-20x48. It does not support INT2. It uses API

```
inv_icm20948_identify_interrupt(struct inv_icm20948 * s, short *int_read)
```

to identify 4 events

1. A sensor data packet is available in the FIFO
2. A Significant Motion is detected
3. A Step Detection event has happened
4. A Bring to See event has been detected

FIFO – If DMP is enabled, the DMP will control what is being outputted to the FIFO. The DMP will output all requested sensor data, minus pedometer. DMP FIFO data includes

```
ACCEL_SET – calibrated accel sample
GYRO_SET – raw gyro sample available. Calibrated gyro data is calculated by manually removing the bias
from raw data
CPASS_SET – raw compass sample
QUAT6_SET – 6 axis accel and gyro orientation quaternion sample
QUAT9_SET – 9 axis accel, gyro, and mag orientation quaternion sample
GEOMAG_SET – 6 axis accel and mag orientation quaternion sample
CPASS_CALIBR_SET – calibrated compass sample
```


Document Number: AN-000246
Revision: 1.0

```
ACT_RECOG_SET – Basic Activity Classification sample to detect walk, run, bike, still, tilt, and drive
events
FLIP_PICKUP_SET – Pickup event state sample
PED_STEPDET_SET – if step detector and step counter is enabled, there will be INT out for step detect
and a FIFO packet sample. Sample it to determine a step detection, not step count.
```

Each sensor packet will include headers, the sensor data, and a footer. The data can be in different formats for each sensor. The formatting will be covered in this appnote

Pedometer Step Count – the only sensor data that is not outputted in the FIFO is the pedometer step count. The pedometer step count is stored in a specific place in the DMP memory. It can be retrieved through this API

```
int dmp_icm20948_get_pedometer_num_of_steps(struct inv_icm20948 * s, unsigned long *steps)
```

WHERE DO I FIND THE LATEST ICM-20X48 DRIVERS WITH DMP?

The software drivers recommended for the ICM-20648 and ICM-20948 which has the DMP firmware can be found at the TDK-InvenSense website

https://invensense.tdk.com/developers/software-downloads/#smartmotion

As of this document the latest release for the 2 products is the SmartMotion eMD

- DK-20648 SmartMotion eMD 1.0.5
- DK-20948 SmartMotion eMD 1.1.0

THE DMP SEEMS TO ONLY OUTPUT QUATERNIONS FOR SENSOR FUSION? WHAT ABOUT OTHER SENSOR FUSION DATA LIKE EULER ANGLES?

When representing orientation quaternions are the more Mathematical accepted data compared to Euler Angles because of it's inherent issues such as Gimble Lock (search in YouTube "euler gimbal lock explained" if you are interested). Quaternions are also more mathematically complex to calculate compare to Euler Angles.

Therefore the DMP off-loads quaternion calculations from the processor to get the best presentation of orientation. If applications still needs Euler Angles it is a simple calculation to take quaternions to Euler Angles and you will still avoid it's problems using this method.

The eMD has an example in its APIs

```
int inv_icm20948_augmented_sensors_get_orientation(long orientation[3], const long quat9axis_3e[4])
```

which takes the quaternions and converts it to Euler.

THE DMP QUATERNIONS SEEMS TO ONLY BE QX, QY, AND QZ, ISN'T THERE A QW?

This was decided and implemented on purpose mainly to reduce the I2C/SPI traffic when the processor retrieves the quaternions from the DMP FIFO output. In Quaternions –

$$Qw^2 + Qx^2 + Qy^2 + Qz^2 = 1$$

Qw can be calculated easily on the processor from Qx, Qy, and Qz based on this equation.

Another question people ask is each quaternion element is from the range -1 to 1, based on this calculation Qw can only 0 to 1. It terms of quaternions Qw and -Qw which is a scale factor is the same function.

DO I NEED TO USE THE DMP?

No it is not necessary for you to use the DMP. Many customers use the ICM-20948 as a standalone 9-axis sensor. Many customers do not need sensor fusion or the gestures and implement their own algorithms. So it is not necessary to use the DMP to use the accel, gyro, or mag data.

WHAT ARE THE DIFFERENCES BETWEEN ICM-20648 AND ICM-20948 DMPS?

There are no differences between the DMPs of the ICM-20648 and ICM-20948. It is the same DMP hardware version (ver 3a) and therefore the same DMP image can run on both. The ICM-20948 is a ICM-20648 with an AKM9916 connected via the AUX-I2C pins in the same package. Therefore the same exact DMP image in the eMDs with the same exactly features are on both.

WHERE ARE THE DMP DOCUMENTATIONS LIKE THE DMP REGISTER MAPS? THE DATASHEETS DOES NOT SPECIFY THE DMP REGISTERS.

This question comes up a lot. In the ICM-20948 and ICM-20648 datasheets, the only DMP registers are bits that enables/disables, resets, and interrupt output. Not much in how to enable certain sensor outputs or features. The reason is because of the nature of the DMP. The DMP as stated before is a FPGA-like processor which means the features are dependent on the image that is being downloaded and the "registers" are just indexes in the DMP memory. This also means when there are changes or updates to the DMP image, these "registers" are subject to change as well as the features.

Therefore the DMP features and it's configurations is considered to be firmware while only the hardware registers are defined in the datasheets. That means DMP settings such as ODR, which sensors to enable and disable, and the outputs are accessed through the DMP APIs of that particular DMP image.

The DMP APIs are defined in Icm20948Dmp3Driver.h or the similar file for the ICM-20648. You will see API function defines such as

```
int inv_icm20948_load_firmware(struct inv_icm20948 * s, const unsigned char *dmp3_image, unsigned int dmp3_image_size);
void inv_icm20948_get_dmp_start_address(struct inv_icm20948 * s, unsigned short *dmp_cnfg);
int dmp_icm20948_reset_control_registers(struct inv_icm20948 * s);
int dmp_icm20948_set_data_output_control1(struct inv_icm20948 * s, int output_mask);
int dmp_icm20948_set_data_output_control2(struct inv_icm20948 * s, int output_mask);
int dmp_icm20948_set_data_interrupt_control(struct inv_icm20948 * s, uint32_t interrupt_ctl);
int dmp_icm20948_set_FIFO_watermark(struct inv_icm20948 * s, unsigned short fifo_wm);
……
```

You will see the DMP Memory Indexes for the DMP "registers" are defined in the Icm20948Dmp3Driver.c.

```
// data output control
#define DATA_OUT_CTL1       (4 * 16)
#define DATA_OUT_CTL2       (4 * 16 + 2)
#define DATA_INTR_CTL       (4 * 16 + 12)
#define FIFO_WATERMARK      (31 * 16 + 14)

// motion event control
#define MOTION_EVENT_CTL    (4 * 16 + 14)

// indicates to DMP which sensors are available
/* 1: gyro samples available
2: accel samples available
8: secondary samples available */
#define DATA_RDY_STATUS       (8 * 16 + 10)

……
```

CAN DMP FIRMWARE FOR MPU-6500/MPU-9250 BE PORTED TO ICM-20X48?

Unfortunately no. The DMP in the ICM-20648 and ICM-20948 is the hardware version DMP 3a. These 2 DMPs are compatible. However MPU motion devices are DMP 2 hardware which are not compatible.

IS THE DMP CODE PRELOADED ON THE ICM?

The ICM motion MEMS devices does not contain any non-volatile memory. Therefore, the DMP image needs to be loaded to the DMP memory on the ICM-20x48 upon power on if customers want to use the DMP. As long as power is provided to the ICM-20x48 the DMP firmware will stay in memory. However if power is removed the DMP image will need to be reloaded again.

WHERE IS THE DMP CODE IN THE DRIVER?

The DMP firmware is a blob of hex codes that needs to be downloaded to the ICM-20x48 DMP memory. In the eMD driver the DMP image is contained in the file “icm20948_img.dmp3a.h” in the directory

eMD-SmartMotion-ICM20948-1.1.0-MP\EMD-App\src\ICM20948

The ICM-20648 has the similar file and directory.

WHAT IS THE STARTING ADDRESS OF THE DMP? WHAT IS THE CODE SIZE OF THE DMP?

Both of these can be found in the eMD driver file Icm20948Dmp3Driver.c

```
#define DMP_START_ADDRESS   ((unsigned short)0x1000)
#define DMP_CODE_SIZE 14301
```

The DMP_CODE_SIZE is in bytes do you can see that the DMP firmware that needs to be loaded will be around 14kB.

CAN I CUSTOMIZE OR REQUEST A NEW DMP FEATURE? CAN I CONFIGURE THE GESTURES OR SENSOR FUSION?

We get this request a lot. Customers will request a new DMP feature or to detect a new gesture. We also get requests to change the sensor fusion to make it more sensitive or faster convergence, etc…

Unfortunately, the answer is usually no we cannot do a special customization of the DMP. The current DMP firmware was developed and tested and passes all Android certification related to its sensors. Developing and testing new DMP features and firmware requires extensive resources and time and unless there are significant business reasons, TDK-InvenSense usually cannot prioritize the resources. Unfortunately, there are no IDE or tools we can provide customers either to develop DMP code themselves either.

HOW FAST IS THE DMP AND IT’S SENSOR FUSION?

The DMP3a Hardware which is embedded in the ICM-20648 and ICM-20948 runs at 20Mhz speed. The estimated DMP instruction set for 9-axis sensor fusion is 25000. The estimated delay of the DMP sensor fusion is ~1.2ms.

WHAT IS THE PERFORMANCE OF THE SENSOR FUSION AND GESTURES DETECTION?

The performance of the sensor fusion and gesture detection is more related to the sensor hardware specs instead of the DMP algorithms. If your sensor’s noise, offsets, temperature drift, and sampling rate is good your sensor fusion and gesture detection will usually be very good.

The driver passes all Android certification for sensor fusion, pedometer functions, significant motion, and basic activity detection. All these are functions the DMP processes.

In general, the sensor fusion performance is estimated at

Dmp 6x:
Static ~ +-1 deg
Normal motion ~ +-3 deg

Dmp9x without mag disturbance
Static ~ +-2 deg
Normal motion ~ +-5 deg

AT WHAT ODR IS THE SENSOR FUSION?

The range of ODR for DMP sensor fusion output is from 50Hz to 225Hz max. The sensors itself can have much higher ODR up to 1Khz and beyond, but the DMP sensor fusion is limited due to the hardware.

WHAT ORIENTATION COORDINATE SYSTEM DOES THE DMP PROCESS DATA? CHIP OR BODY/WORLD COORDINATE FRAME?

While the DMP has APIs to put in an orientation matrix which will convert the sensor data from chip coordinate system to body/world coordinate system, the DMP is setup to processes sensor data in chip frame. This means that the orientation matrix in the DMP is the identity matrix.

The only exception is that there is a matrix for the gesture Bring-To-See which the DMP will use to detect this gesture signature. This will use the API

```
int dmp_icm20948_set_B2S_matrix(struct inv_icm20948 * s, int *b2s_mtx);
```

Another note is that because the compass die is separate from the accel/gyro die internally in the ICM-20948. The compass matrix in the DMP will need to align with the identity matrix of the accel and gyro. Even though the compass matrix will not be identity matrix, because the die is a fixed alignment with the accel and gyro, the driver will know which matrix to apply and this is automatically done in the SmartMotion eMD API

```
void INV_EXPORT inv_icm20948_register_aux_compass(struct inv_icm20948 * s,
   enum inv_icm20948_compass_id compass_id, uint8_t compass_i2c_addr);
```

if you are manually aligning the mag up with the accel and gyro chip frame. Then you will need to this API to set the compass orientation matrix

```
int dmp_icm20948_set_compass_matrix(struct inv_icm20948 * s, int *compass_mtx);
```

The chip to body/world frame conversion is done after the data is retrieved from the DMP FIFO outside of the processing itself in the driver. To make sure you chip to body frame matrix is setup, you want to make sure in sensor.c the structure

```
/*
* Mounting matrix configuration applied for Accel, Gyro and Mag
*/

static const float cfg_mounting_matrix[9]= {
       1.f, 0, 0,
       0,  1.f, 0,
       0, 0, 1.f
};
```

Is applied correctly.

HOW TO ENABLE THE DMP WATERMARK FEATURE?

The ICM-20x48 has a hardware watermark in which can be programmed through the registers found in the datasheets. This watermark will trigger an interrupt when the FIFO fills to 50% capacity. The typical FIFO configuration is 1024 bytes.

However the hardware watermark cannot be enabled if the DMP is enabled. This is because the you want to use the DMP, the DMP will need to control the interrupts and also the FIFO output.

There is a DMP watermark API

```
int dmp_icm20948_set_FIFO_watermark(struct inv_icm20948 * s, unsigned short fifo_wm);
```

which the firmware sets the DMP watermark at 80% of the FIFO.

The watermark feature though is only enabled if batching is enabled which is an Android phone feature. Because of this in the default SmartMotion SW this batching is not enabled.

HOW DO I INTEGRATE THE DMP? WHAT IS THE BEST STARTING POINT?

Start with the eMD code. The eMD code provides a complete driver on interfacing with the DMP and is the best place to start. Customer can also get a DK-20648 or DK-20948 evaluation kit and run the eMD driver on it and step through the code. Eval kits are available for purchase at Mouser, Arrow, CDI, and DigiKey. The eMD runs on a G55 Atmel MCU in which the toolchain is free to download.

DMP PROCEDURE

This section will use the eMD-SmartMotion-ICM20948-1.1.0-MP as the basis of how the DMP is initialized, configured, and how to get data from it. The procedure and flow are very similar to the eMD-SmartMotion_ICM20648_1.0.5 except for obvious naming conventions (instead of 20648 it is 20948). This section will apply to both.

BEFORE THE DMP – SETTING UP THE ICM HARDWARE

Before the DMP can be initialize, make sure the sensor hardware is setup completely. This can include

- Configure clock source through `PWR_MGMT_1`.
- Enable accel and gyro sensors through `PWR_MGMT_2`.
- Configure Gyro/Accel in Low power mode with `LP_CONFIG`.
- Set Gyro FSR (Full scale range) to 2000dps through `GYRO_CONFIG_1`.
- Set Accel FSR (Full scale range) to 4g through `ACCEL_CONFIG`.
- Enable interrupt for FIFO overflow from FIFOs through `INT_ENABLE_2`.
- Turn off what goes into the FIFO through `FIFO_EN, FIFO_EN_2`.
- Turn off data ready interrupt through `INT_ENABLE_1`.
- Reset FIFO through `FIFO_RST`.

In the eMD code the ICM hardware and the DMP initialization are combined together in the API

```
/** Should be called once on power up. Loads DMP3, initializes internal variables needed
*   for other lower driver functions.
*/
int inv_icm20948_initialize_lower_driver(struct inv_icm20948 * s, enum SMARTSENSOR_SERIAL_INTERFACE type, const uint8_t *dmp3_image, uint32_t
dmp3_image_size)
```

We will go into more details on the DMP initialization. If you follow this API it will contain everything you need to get the ICM part and the DMP up and running.

The DMP is optimized for the accel and gyro to be in duty cycle mode to save power.

INITIALIZING THE DMP

Load the DMP Image

The DMP firmware size is ~14Kbytes and is located in the file icm20948_img.dmp3a.h. DMP program code may be loaded one byte at a time or in burst mode as explained below.

- Load Firmware One Byte at a Time

Starting from byte [0] of the firmware image provided, perform the following steps to load each byte of image to Nth byte of DMP address. N must start from address (0x90) and increase accordingly:

- Write the value (N >> 8) to `MEM_BANK_SEL`
- Write the value (N & 0xFF) to `MEM_START_ADDR`
- Write the data to `MEM_R_W` starting from the 0x90th byte of DMP firmware image.

Repeat the steps above for each byte of the DMP firmware image.

- Load Firmware up to 256 Bytes at a Time

Up to 256 bytes can be burst written. Please follow the steps outlined in previous section. However, the maximum number of bytes that can be written at once is not always 256, but is given by (256 – (N & 0xFF)).

eMD Driver Example

In the eMD, the DMP image is loaded 256 bytes at a time. After the DMP is loaded, the driver reads out of image from the DMP memory and confirms that it is correct.

```
/** @brief Loads the DMP firmware from SRAM
* @param[in] data  pointer where the image
* @param[in] size  size if the image
* @param[in] load_addr  address to loading the image
* @return 0 in case of success, -1 for any error
*/
int inv_icm20948_firmware_load(struct inv_icm20948 * s, const unsigned char *data_start, unsigned short size_start, unsigned short load_addr)
{
   int write_size;
   int result;
   unsigned short memaddr;
   const unsigned char *data;
   unsigned short size;
   unsigned char data_cmp[INV_MAX_SERIAL_READ];
   int flag = 0;

      if(s->base_state.firmware_loaded)
           return 0;

   // Write DMP memory
   data = data_start;
   size = size_start;
   memaddr = load_addr;
   while (size > 0) {
      write_size = min(size, INV_MAX_SERIAL_WRITE);
      if ((memaddr & 0xff) + write_size > 0x100) {
         // Moved across a bank
         write_size = (memaddr & 0xff) + write_size - 0x100;
      }
      result = inv_icm20948_write_mems(s, memaddr, write_size, (unsigned char *)data);
      if (result)
         return result;
      data += write_size;
      size -= write_size;
      memaddr += write_size;
   }

   // Verify DMP memory

   data = data_start;
   size = size_start;
   memaddr = load_addr;
   while (size > 0) {
      write_size = min(size, INV_MAX_SERIAL_READ);
      if ((memaddr & 0xff) + write_size > 0x100) {
         // Moved across a bank
         write_size = (memaddr & 0xff) + write_size - 0x100;
      }
      result = inv_icm20948_read_mems(s, memaddr, write_size, data_cmp);
      if (result)
         flag++; // Error, DMP not written correctly
      if (memcmp(data_cmp, data, write_size))
         return -1;
      data += write_size;
      size -= write_size;
      memaddr += write_size;
   }

#if defined(WIN32)
   //if(!flag)
```


Document Number: AN-000246
Revision: 1.0

```
    // inv_log("DMP Firmware was updated successfully..\r\n");
#endif

   return 0;
}
```

Setup the DMP Starting Address, FIFO, DMP_EN in the Hardware Registers

After loading the DMP, these steps is needed to make sure the DMP will run.

The DMP Start Address which is provided in the file icm20948Dmp3Driver.c needs to be written to the PRGM_STRT_ADDRH/PRGM_STRT_ADDRL registers. The code is available is this API.

- `#define DMP_START_ADDRESS   ((unsigned short)0x1000)`
- `int inv_icm20948_set_dmp_address(struct inv_icm20948 * s)`

The FIFO and the DMP needs to enable

- `s->base_state.user_ctrl |= BIT_DMP_EN | BIT_FIFO_EN;`
- `result |= inv_icm20948_write_single_mems_reg(s, REG_USER_CTRL, s->base_state.user_ctrl);`

DMP interrupts need to be enabled

- `data = 0x2;`
- `result |= inv_icm20948_write_mems_reg(s, REG_INT_ENABLE, 1, &data); // Enable DMP Interrupt`

After these 2 steps the DMP should be up and running. However you would not be able to get data until you enable a sensor output.

Setup DMP FSR, Mounting Matrix, FIFO Watermark, initial sample rates

These are optional setups for the DMP. This is because there is a default value already. But it is recommended to go through these configurations to get correct DMP data output.

- The default eMD Full Scale Range is set to +-4G for accel and +-2000dps for gyro. If need to change please change it here.

```
int inv_icm20948_init_scale(struct inv_icm20948 * s)
{
   /* Force accelero fullscale to 4g and gyr to 2000dps */
   inv_icm20948_set_accel_fullscale(s, MPU_FS_4G);
   inv_icm20948_set_gyro_fullscale(s, MPU_FS_2000dps);

   return 0;
}
```

- There are 3 mounting matrixes that the DMP will use in its processing
  - Accel and Gyro does not need to be configured because the DMP will always use identity matrix for these sensor. This means that any mounting matrix transformations will be done post DMP. The eMD has this already setup and all you need to change is the mounting matrix here

```
/*
* Mounting matrix configuration applied for Accel, Gyro and Mag
*/
static const float cfg_mounting_matrix[9]= {
1.f, 0, 0,
0,  1.f, 0,
0, 0, 1.f
};
```

- The compass mounting matrix in the DMP will need to be configured if you have a compass. For the ICM-20948 the eMD driver will automatically set this since because the accel, gyro, and mag are in one package we know what the compass matrix will be. This is set in the compass setup

```
inv_icm20948_register_aux_compass(&icm_device, INV_ICM20948_COMPASS_ID_AK09916,
AK0991x_DEFAULT_I2C_ADDR);
```

If you are using the ICM-20648 and have no compass you will skip this. If you attached an external mag through the AUX I2C pins of the ICM-20648, you will need to initialize the compass matrix in the above API specifically you might need specify the following for the new compass.

```
case INV_ICM20948_COMPASS_ID_AK08963:
   s->secondary_state.compass_slave_id = HW_AK8963;
   s->secondary_state.compass_chip_addr = compass_i2c_addr;
   s->secondary_state.compass_state = INV_ICM20948_COMPASS_INITED;
   /* initialize mounting matrix of compass to identity akm8963 */
   s->mounting_matrix_secondary_compass[0] = 1;
   s->mounting_matrix_secondary_compass[4] = 1;
   s->mounting_matrix_secondary_compass[8] = 1;
   break;
```

- The only DMP Matrix that will need to relay on `cfg_mounting_matrix[9]` is the matrix for B2S which is the “Bring to See” feature in the DMP mainly for wrist worn devices such as smartwatches. In the eMD the API

```
/**
* Sets B2S accel orientation matrix to DMP.
* @param[in] b2s_mtx. Unit: 1 = 2^30.
*/
int dmp_icm20948_set_B2S_matrix(struct inv_icm20948 * s, int *b2s_mtx)
```

is called and it is passed in the `cfg_mounting_matrix[9]` to set the B2S matrix.

- FIFO watermark is set in the eMD at 80%. This will reduce I2C traffic so that allows the external processor to do a read burst once the FIFO (1Kb) reaches 80% full. A FIFO interrupt will be outputted to the processor once that threshold is reached to indicate the watermark. This is done by calling this API

```
// set FIFO watermark to 80% of actual FIFO size
result |= dmp_icm20948_set_FIFO_watermark(s, 800);
```

- Initial Sample Rates for B2S and BAC to 56Hz. These 2 features needs a minimum of 56Hz in order to more accurately detect these gestures. Therefore it is recommended that we make sure that if these 2 features are enabled that the minimum ODR necessary is 56Hz. This is called through these APIs

```
// Init the sample rate to 56 Hz for BAC,STEPC and B2S
dmp_icm20948_set_bac_rate(s, DMP_ALGO_FREQ_56);
dmp_icm20948_set_b2s_rate(s, DMP_ALGO_FREQ_56);
```

Again these configurations while not necessary for the DMP to run, is recommended to make sure the DMP processes the data accurately.

ENABLING A SENSOR OUTPUT IN THE DMP

Now that you should have the DMP image loaded and everything ready to go, it is now to enable a DMP sensor and get it’s output.

Android and DMP Sensors

The eMD was developed with Google Android criteria and requirements as a goal. While there are wide range types of devices with motion, Android has a wide market and allows the eMD to be application to the most customers without TDK-InvenSense having to develop a new software for each market segments.

At the time of the eMD development the Android iteration was up to ‘L’. Luckily as of Android ‘R’ or 11 release, there has been little change to the sensor requirements so the eMD is still very much applicable to current devices.

The ICM-20x48 leverages the DMP to do the majority of the sensor fusion that Android requires. These are the possible sensor data requests in Android however not all of these are supported in the eMD for obvious reasons, like heart rate etc…

```
/* enum for android sensor*/
enum ANDROID_SENSORS {
    ANDROID_SENSOR_META_DATA = 0,
    ANDROID_SENSOR_ACCELEROMETER,
    ANDROID_SENSOR_GEOMAGNETIC_FIELD,
    ANDROID_SENSOR_ORIENTATION,
    ANDROID_SENSOR_GYROSCOPE,
    ANDROID_SENSOR_LIGHT,
    ANDROID_SENSOR_PRESSURE,
    ANDROID_SENSOR_TEMPERATURE,
    ANDROID_SENSOR_WAKEUP_PROXIMITY,
    ANDROID_SENSOR_GRAVITY,
    ANDROID_SENSOR_LINEAR_ACCELERATION,
    ANDROID_SENSOR_ROTATION_VECTOR,
    ANDROID_SENSOR_HUMIDITY,
    ANDROID_SENSOR_AMBIENT_TEMPERATURE,
    ANDROID_SENSOR_MAGNETIC_FIELD_UNCALIBRATED,
    ANDROID_SENSOR_GAME_ROTATION_VECTOR,
    ANDROID_SENSOR_GYROSCOPE_UNCALIBRATED,
    ANDROID_SENSOR_WAKEUP_SIGNIFICANT_MOTION,
    ANDROID_SENSOR_STEP_DETECTOR,
    ANDROID_SENSOR_STEP_COUNTER,
    ANDROID_SENSOR_GEOMAGNETIC_ROTATION_VECTOR,
    ANDROID_SENSOR_HEART_RATE,
    ANDROID_SENSOR_PROXIMITY,

    ANDROID_SENSOR_WAKEUP_ACCELEROMETER,
    ANDROID_SENSOR_WAKEUP_MAGNETIC_FIELD,
    ANDROID_SENSOR_WAKEUP_ORIENTATION,
    ANDROID_SENSOR_WAKEUP_GYROSCOPE,
    ANDROID_SENSOR_WAKEUP_LIGHT,
    ANDROID_SENSOR_WAKEUP_PRESSURE,
    ANDROID_SENSOR_WAKEUP_GRAVITY,
    ANDROID_SENSOR_WAKEUP_LINEAR_ACCELERATION,
    ANDROID_SENSOR_WAKEUP_ROTATION_VECTOR,
    ANDROID_SENSOR_WAKEUP_RELATIVE_HUMIDITY,
    ANDROID_SENSOR_WAKEUP_AMBIENT_TEMPERATURE,
    ANDROID_SENSOR_WAKEUP_MAGNETIC_FIELD_UNCALIBRATED,
    ANDROID_SENSOR_WAKEUP_GAME_ROTATION_VECTOR,
    ANDROID_SENSOR_WAKEUP_GYROSCOPE_UNCALIBRATED,
    ANDROID_SENSOR_WAKEUP_STEP_DETECTOR,
    ANDROID_SENSOR_WAKEUP_STEP_COUNTER,
    ANDROID_SENSOR_WAKEUP_GEOMAGNETIC_ROTATION_VECTOR,
    ANDROID_SENSOR_WAKEUP_HEART_RATE,
    ANDROID_SENSOR_WAKEUP_TILT_DETECTOR,
    ANDROID_SENSOR_RAW_ACCELEROMETER,
    ANDROID_SENSOR_RAW_GYROSCOPE,
    ANDROID_SENSOR_NUM_MAX,

     ANDROID_SENSOR_B2S,
    ANDROID_SENSOR_FLIP_PICKUP,
    ANDROID_SENSOR_ACTIVITY_CLASSIFICATON,
    ANDROID_SENSOR_SCREEN_ROTATION,
    SELF_TEST,
    SETUP,
    GENERAL_SENSORS_MAX

};
```

To see the actual sensors supported you can look at the structure

```
const short inv_androidSensor_to_control_bits[ANDROID_SENSOR_NUM_MAX]
```

in

```
static int inv_enable_sensor_internal(struct inv_icm20948 * s, unsigned char androidSensor, unsigned char enable, char * mems_put_to_sleep)
```

you will see then that for the ICM-20648 and ICM-20948 these sensor data output are supported

- SENSOR_ACCELEROMETER
- SENSOR_MAGNETOMETER
- SENSOR_GYROSCOPE
- SENSOR_GRAVITY
- SENSOR_LINEAR_ACCELERATION
- SENSOR_ROTATION_VECTOR
- SENSOR_UNCAL_MAGNETOMETER
- SENSOR_GAME_ROTATION_VECTOR
- SENSOR_UNCAL_GYROSCOPE
- SENSOR_SMD
- SENSOR_STEP_DETECTOR
- SENSOR_STEP_COUNTER
- SENSOR_GEOMAG_ROTATION_VECTOR
- SENSOR_TILT_DETECTOR
- SENSOR_PICK_UP_GESTURE
- SENSOR_BAC
- SENSOR_B2S
- SENSOR_RAW_ACCELEROMETER
- SENSOR_RAW_GYROSCOPE

Setting DMP ODR

Setting the correct DMP ODR is probably one of the most complex algorithms in the eMD. This is because you can set the accel, gyro, and mag set to different ODRs as well as the sensor fusion and gestures at different ODRs. If you have more than 1 sensor output requested the algorithm needs to go through each request, see which raw sensors are needed and adjust accordingly.

For example if you requested RV@225Hz, gyro @ 50Hz, 6-axis RV@ 112Hz. It will take each request…see which sensors are enabled currently and adjust the accel, gyro, and mag sampling rate accordingly so that all sensor data requested can be fulfilled. In this case, RV which is accel, gyro, and mag @ 225Hz will be the highest requirement so the sensor samples rates will need to be at 225Hz.

Because of the complexity the recommendation is that you keep it simple to 1-2 sensors at same ODR. Or you let the eMD driver implement the decision for you.

If you are ok with the default 102Hz ODR then you can actually not worry about setting the ODR.

Simple Implementation – follow the code

Using the eMD driver, the procedure to setup the correct ODR is the following -

- `inv_icm20948_set_sensor_period(&icm_device, idd_sensortype_conversion(sensor), edata->d.command.period / 1000)-> inv_icm20948_set_odr(s, androidSensor, period)->`
    - `inv_icm20948_set_odr will calculate which ODR is necessary on which sensors are enabled after calculations…`
- `inv_icm20948_set_odr-> inv_set_hw_smplrt_dmp_odrs()`
    - `inv_set_hw_smplrt_dmp_odrs will enter into correct modes and also set DMP output ODR in DividerRateSet()`
- `inv_icm20948_set_odr-> inv_icm20948_set_gyro_sf`
    - `sets the gyro rates for quaternions`

In-Depth ODR Calculations

Here are the basic rules for DMP ODR

- BAC, Step Counter, Tilt, Flip, pickup, B2S needs minimum 56Hz
- GRV and RV has 50Hz minimum up to 225Hz
- All other sensors (accel, gyro, mag) can be 1Hz to 200Hz

to set the ODR you need to do the following –

1. Hardware Registers for ACCEL_SMRT_DIV and GYRO_SMRT_DIV –
    a. The formula of course is 1125Hz/(reg_val-1) = HW ODR

2. GRV and RV is limited to boundary of 50Hz to 225Hz ODR
3. Accel APIs to setup the hardware registers

```
result |= inv_icm20948_ctrl_set_accel_quaternion_gain(s, hw_smplrt_divider);
result |= inv_icm20948_ctrl_set_accel_cal_params(s, hw_smplrt_divider);
          result |= inv_icm20948_set_accel_divider(s, hw_smplrt_divider - 1);
```

4. Gyro APIs to setup the hardware registers

```
          result |= inv_icm20948_set_gyro_divider(s, (unsigned char)(hw_smplrt_divider -
          1));
```

5. DMP output can only be a phase of the input ODR delay. Meaning for example if you have ODR of 225Hz DMP for GRV, you want a lower ODR for accel and gyro say like 50Hz.
    a. The GRV delay in ms is 1/225 = ~4.44ms delay
    b. Targeted accel and gyro is 50Hz so 1/50 = 20ms delay
    c. Therefore the closest accel and gyro would be ~22.2ms delay since 4.44 * 5 = 22.2 -> ~45Hz
6. DMP_DIV is set in API `int dmp_icm20948_set_sensor_rate(struct inv_icm20948 * s, int invSensor, short divider)`
7. Everything time ODR is set this API is called immediately afterwards to setup the gyro full scale range in the DMP
`inv_icm20948_set_gyro_sf(s, inv_icm20948_get_gyro_divider(s), inv_icm20948_get_gyro_fullscale(s));`

This has always been the most complex part of the code. I will go through example application

Example target - 50Hz GRV, 50Hz Accel, 50Hz Gyro (output packets from DMP will be same every interrupt)

- The closest value for the register value for accel and gyro is '21' because 1125/(21 + 1) = ~51.136Hz. Therefore hw_smplrt_divider = 21
    - `inv_icm20948_ctrl_set_accel_quaternion_gain(s, hw_smplrt_divider);`
    - `inv_icm20948_ctrl_set_accel_cal_params(s, hw_smplrt_divider);`
    - `inv_icm20948_set_accel_divider(s, hw_smplrt_divider - 1);`
    - `inv_icm20948_set_gyro_divider(s, (unsigned char)(hw_smplrt_divider - 1));`
- Because accel and gyro are also 50Hz, then there is no DMP divider necessary because it is in exact phase as the ODR input. Therefore divider = '0' for both accelID (0) and gyro(5)
    - `dmp_icm20948_set_sensor_rate(struct inv_icm20948 * s, int invSensor, short divider)`

Enabling the Sensor

The eMD actually makes enabling the sensors fairly easy.

i. `inv_icm20948_enable_sensor(&icm_device, idd_sensortype_conversion(sensor), 1)-> inv_icm20948_ctrl_enable_sensor(s, androidSensor, state)-> inv_enable_sensor_internal(s, androidSensor, enable, &s->mems_put_to_sleep)`
ii. `inv_enable_sensor_internal` will be main function which will enable the sensors, enable the interrupts, and recalculate the ODRs

Stopping the Sensor

Same steps as enable sensors except call to

`inv_icm20648_enable_sensor(struct inv_icm20648 * s, enum inv_icm20648_sensor sensor, inv_bool_t state)`

set the 'state' variable to 0

DMP OUTPUT – FIFO AND INTERRUPTS

The DMP outputs all sensor data (besides pedometer step count - see FAQ section) to the FIFO. Each sensor packet has headers and also the sensor data in a particular format and length. How the SmartMotion eMD handles the DMP FIFO starts in

`int inv_icm20948_poll_sensor(struct inv_icm20948 * s, void * context, void (*handler)(void * context, enum inv_icm20948_sensor sensor, uint64_t timestamp, const void * data, const void *arg))`

this API will do the following

Document Number: AN-000246
Revision: 1.0

1. Identity Interrupt – identity which interrupt and therefore which event you are processing. There are basically 4 types of interrupts
    - A sensor data packet is available in the FIFO
    - A Significant Motion is detected
    - A Step Detection event has happened
    - A Bring to See event has been detected
2. Get the timestamp – the timestamp is MCU based not from the DMP. Timestamps are in microsecond units.
3. Process FIFO – pop off headers the process the sensor data accordingly. Keep going until all full sensor data in FIFO has been processed
4. Convert Data Format – Convert from DMP format, convert to full scale range, convert from chip to body frame, apply timestamps.

Detailed DMP FIFO output format –

The DMP deals with a lot of floating-point calculations. To provide better accuracy and avoid truncating, the calculations are done in Q format. **Q** (number) **format** is a fixed-point method of coding fractional and whole integers for processing by a computer's CPU or a digital signal processor (**DSP**). The **Q format** is used to enable rational number processing by a standard integer hardware arithmetic logic unit.

To convert from Q format to it's original units. Customers will need to take the Q formatted number and divide by the 2^N where N is the number of bits used to designate the fractional portion of the number. For example, if a number X is in Q16 format, you will take X and divide by 2^16 and that will get you the original number. You should note that you will still need to understand what the original units were.

Here are the data formatted packets the DMP outputs -

Accel Sensor Data Packet: 10 bytes

- Header Packet : 2 bytes (`0x8000`)
- Accel Sensor Data: 6 bytes
    - Accel X, Y, Z in Q25 format
- Footer Packet : 2 bytes

Note – data used for raw accel (unscaled accel values), accel (scaled accel), and linear accel (scaled accel minus gravity). There is only 1 accel value in the DMP which can be calibrated or uncalibrated depending on the calibration algorithms.

Gyro Sensor Data Packet: 16 bytes (`0x4000`)

- Header Packet : 2 bytes
- Gyro Raw Sensor Data: 6 bytes
    - Gyro X, Y, Z in Q15 format
    - In scaled hardware units depending on FSR setting
- Gyro Bias Data: 6 bytes
    - Gyro X, Y, Z, in Q20 format
    - Original units always in +-2000dps scale
- Footer Packet : 2 bytes

Note – used for gyroscope and uncalibrated gyroscope. Gyroscope is calibrated output and the driver will manually remove the bias from the raw.

Raw Compass Sensor Data Packet: 10 bytes

- Header Packet : 2 bytes (`0x2000`)

- Compass Raw Sensor Data: 6 bytes
    - Compass X, Y, Z in Q16 format
- Footer Packet : 2 bytes

Note – mag biases are read from DMP memory indexes and are not outputted in FIFO.

Calibrated Compass Sensor Data Packet: 16 bytes

- Header Packet : 2 bytes (`0x0020`)
- Compass Calibrated Sensor Data: 12 bytes
    - Compass data X, Y, Z in Q16 format
- Footer Packet : 2 bytes

6-Axis (GRV) Accel, Gyro Quaternion Sensor Data Packet: 16 bytes

- Header Packet : 2 bytes (`0x0800`)
- 6-Axis AG Quaternion Sensor Data: 12 bytes
    - Quaternions X, Y, Z in Q30 format
- Footer Packet : 2 bytes

Note – Quaternion W is calculated in driver (see FAQ). Gravity vector and also linear acceleration is dependent on 6-axis quaternions being calculated.

9-Axis (RV) Accel, Gyro, Mag Quaternion Sensor Data Packet: 18 bytes

- Header Packet : 2 bytes (`0x0400`)
- 9-Axis AGM Quaternion Sensor Data: 12 bytes
    - Quaternion X, Y, Z in Q30 format
- Quaternion Estimated Error: 2 bytes
    - In Q29 format. 9-axis does not have accuracy flag but has a floating point estimated angle error calculated from DMP.
- Footer Packet : 2 bytes

Note – Euler angles from deprecated ORIENTATION sensor can be calculated from the 9-axis RV and there is example in the code.

6-Axis (GeoMag) Accel, Mag Quaternion Sensor Data Packet: 18 bytes

- Header Packet : 2 bytes (`0x0100`)
- 6-Axis AM Quaternion Sensor Data: 12 bytes
    - Quaternion X, Y, Z in Q30 format
- Quaternion Estimated Error: 2 bytes
    - In Q29 format. GeoMag does not have accuracy flag but has a floating point estimated angle error calculated from DMP.
- Footer Packet : 2 bytes

Gesture – Pedometer Step Detect: 8 bytes

- Header Packet : 2 bytes
- Pedometer Timestamp Data: 4 bytes
- Footer Packet : 2 bytes

Document Number: AN-000246
Revision: 1.0

Note – this packet from DMP is only timestamp. The actual pedometer step count will be read from DMP memory.

DMP FIFO Special Cases - Sensor Accuracy Flags, BAC, and Pickup/Flip

These sensor data are special cases in which the DMP can output the data packets by itself or attached it to an existing sensor data. For example, if the accuracy flag of the accel changes and a GRV sensor data is being outputted. The typically GRV data format will be this

Typical 6-Axis (GRV) Accel, Gyro Quaternion Sensor Data Packet: 16 bytes

- Header Packet : 2 bytes
- 6-Axis AG Quaternion Sensor Data: 12 bytes
- Footer Packet : 2 bytes

However if a sensor accuracy flag, BAC, or Pickup state changes it can attached itself to the existing sensor data. So if the accel accuracy flag changes it will add

6-Axis (GRV) Accel, Gyro Quaternion Sensor Data Packet with Accel Accuracy Flag: 20 bytes

- Header Packet : 4 bytes
  - 2 extra header 2 bytes to indicate which extra data is included. Can be 1 or more extra data
- 6-Axis AG Quaternion Sensor Data: 12 bytes
- Accel Accuracy Flag Data: 2 bytes
- Footer Packet : 2 bytes

The accuracy flags will be attached to an existing sensor data. Each flag is 2 bytes of data. However BAC and Pickup can be outputted as a sensor data packet by itself if need to. In these cases, the standalone sensor data format would be these.

Gesture – Basic Activity Classification: 12 bytes

- Header Packet : 4 bytes
  - First 2 byte bit mask will mark a header 2 packet is present (`0x0008`)
  - `Header 2 bytes (0x0080)`
- BAC Sensor Data: 2 bytes
  - Indicates the beginning of a new BAC state: Walking, Running, Biking, Still, Tilt, Vehicle
- BAC Timestamp: 4 bytes
- Footer Packet : 2 bytes

Gesture – Flip or Pickup Detect: 8 bytes

- Header Packet : 4 bytes
  - First 2 byte bit mask will mark a header 2 packet is present (`0x0008`)
  - `Header 2 bytes (0x0400)`
- Flip_Pickup Sensor Data: 2 bytes
- Footer Packet : 2 bytes

Document Number: AN-000246
Revision: 1.0

REVISION HISTORY

REVISION DATE	REVISION	DESCRIPTION
10/08/2020	1.0	Initial Release
7/16/2021	1.1	Changed DMP FIFO size to 1Kb

DECLARATION DISCLAIMER

InvenSense believes the environmental and other compliance information given in this document to be correct but cannot guarantee accuracy or completeness. Conformity documents substantiating the specifications and component characteristics are on file. InvenSense subcontracts manufacturing, and the information contained herein is based on data received from vendors and suppliers, which has not been validated by InvenSense.

©2017 InvenSense. All rights reserved.