Software User Guide

For

DK-20948

Dev Kit

TABLE OF CONTENTS

USEFUL LINKS

TDK website:

http://www.InvenSense.com/

https://www.invensense.com

https://www.microchip.com

http://www.microchip.com/developmenttools/productdetails.aspx?partno=atsamg55-xpro

https://www.microchip.com/avr-support/atmel-studio-7

1 OVERVIEW

The purpose of this document is to give an overview of the ICM20948/SAMG55 Developer Kit that will allow users to create an application based on motion sensors. This document may also serve as a quick start guide for the ICM20948 package and its elements, including setup use of the sample applications.

1.1 INTRODUCTION

The ICM20948/SAMG55 Dev Kit is compatible with the Atmel's ATSAMG55-XPRO evaluation kit based on a SAM G55 Cortex™-M4 processor-based microcontrollers. The supported development tools are Atmel Studio and Embedded Debugger. The purpose of this solution is to allow sensor management and algorithm processing by using a standalone microcontroller. The ICM20948/SAMG55 solution is an embedded sensors combo (accelerometer & gyroscope & magnetometer) on chip, easy to integrate for users developing within the wearable and IoT space. The Dev Kit includes a full sensor software solution.

2 HARDWARE PLATFORM

The TDK Dev Kit platform for ICM20948 consists of the following components:

- TDK SAMG55 Dev Kit with onboard ICM20948
- ICM20948 contains AK09916 based Magnetometer (a.k.a. AKM9916)

2.1 ATSAMG55-XPRO SETUP

The TDK SAMG55 Dev Kit includes a SAMG55J19A microcontroller. For more information on this MCU, please refer to Atmel website (see *Useful Links* section above.)

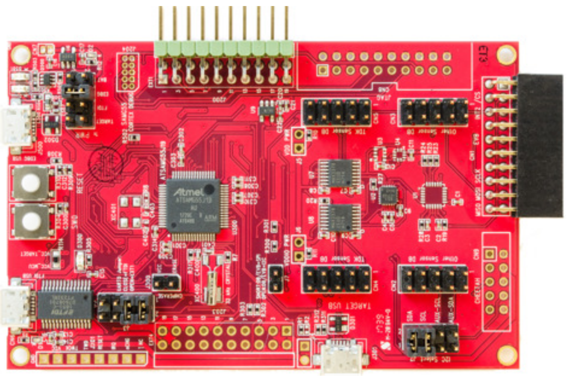


Figure 1 – TDK SAMG55 Dev Kit– Onboard 20948

Figure 1 shows a TDK SAMG55 Dev Kit with an onboard ICM20948 nine axis sensor.

2.2 ATSAMG55 BOARD SETUP

The systems pictured above are configured for I2C communication between the ATSAMG55 and the ICM20948.

Jumper configurations -

- Power (J1)
  - To power the Dev Kit via the EDBG USB port (J500), connect a jumper across pins 3 & 4.
  - To receive power via the FTDI USB port (CN6), connect a jumper across pins 5 & 6 (default).

  Note: The second configuration (power over the FTDI port) is useful when the firmware has already been flashed and no debugging is required. This configuration allows only for only one USB connection, via the FTDI port.

- I2C/SPI configuration (J2)
  - For communication between the ATSAMG55 and the ICM20948 via I2C, add jumpers between pins 1 & 2 and pins 3 & 4.
  - For communication between the ATSAMG55 and the ICM20948 via SPI, remove the jumpers between pins 1 & 2 and pins 3 & 4.

  Note: By default, the firmware and hardware are setup for I2C communication. To use the SPI interface, both the hardware (as described above) and the firmware (as described below) must be changed.

- UART (J3)

- For UART communication over FTDI, connect pins 1 & 2 and pins 3 & 4.

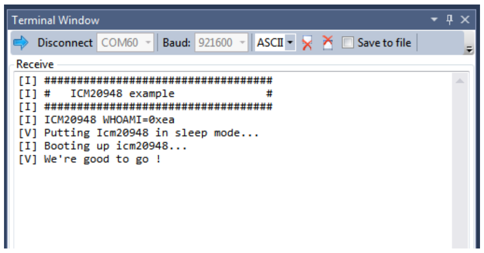
- AKM9916 based Magnetometer
  - The Magnetometer is inbuilt and no external jumper connection is required to access the compass.

2.2.1 Powering the SAMG55 Dev Kit

To power the platform, connect either the J500 or the CN6 port (based on the J1 jumper setting) to a PC using a micro-USB cable.

2.2.2 Debugging on the SAMG55 Dev Kit

To debug or flash the firmware, the EDBG USB port needs to be connected to a host PC using a micro-USB cable. There is also a provision for the firmware to print debug traces on the EDBG UART/USB connector. AtmelStudio Terminal Window can be used to read the messages at baud-rate 921600 by selecting the correct COM port. The serial line configurations are mentioned below. In order to disable data logging across the UART, the INV_MSG_ENABLE macro must be undefined and the firmware rebuilt. If the verbose level needs to be changed, the INV_MSG_ENABLE macro needs to be overloaded with the new desired verbose level.



COM60 Baud: 921600 ASCII

```
[I] ###################################
[I] #   ICM20948 example              #
[I] ###################################
[I] ICM20948 WHOAMI=0xea
[V] Putting Icm20948 in sleep mode...
[I] Booting up icm20948...
[V] We're good to go !
```

Figure 2 – Atmel Studio Terminal Window settings

Speed	921600 bauds
Data bits	8
Stop bits	1
Parity	None
Flow control	None

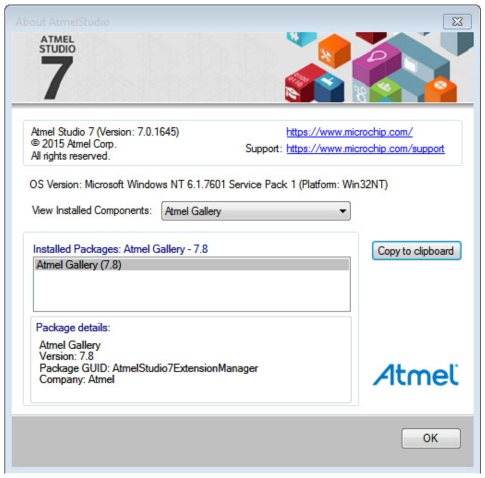
Figure 3 - Serial line configuration

3 *SOFTWARE ENVIRONMENT*

3.1 PREREQUISITE

To build and use the samples application provided as part for the DK-20948 Developer Kit packages, the following software is required:

- An RS232 terminal emulator (such as Putty: http://www.putty.org/)
  - To retrieve traces from provided FW application
- Atmel Studio: http://www.atmel.com/tools/atmelstudio.aspx
  - To load FW binaries and access to the USB EDBG port of the SAMG55 Dev Kit



Atmel Studio 7 (Version: 7.0.1645)
© 2015 Atmel Corp.
All rights reserved.

https://www.microchip.com/
Support: https://www.microchip.com/support

OS Version: Microsoft Windows NT 6.1.7601 Service Pack 1 (Platform: Win32NT)

View Installed Components: Atmel Gallery

Installed Packages: Atmel Gallery - 7.8
Atmel Gallery (7.8)

Package details:
Atmel Gallery
Version: 7.8
Package GUID: AtmelStudio7ExtensionManager
Company: Atmel

Figure 4 - About Atmel Studio

3.2 EMD DEVELOPER KIT TDK PACKAGES

The following package is available:

- *eMD-SmartMotion_ICM20948_x.y.z.zip*

This example targets low performance microcontrollers with a very simple sensor application.

Tools running on the PC for data display are available within the delivered package.

3.3 FW PACKAGE DESCRIPTION

The DK-20948 package includes all the necessary files to create a custom application using an ICM20948 device.

The package is organized as follow

- ***doc*:** Document(s) describing the use of this firmware development platform.
- ***EMD-App*:** contains sample firmware source and project files.
  - ***src:***
    - **At the top level:** Shared .c & .h files.
    - **ASF:** Shared Atmel system files.
    - **config:** Shared config files.
    - **ICM*:** Sensor specific files, main.[c,h], sensor.[c,h] and system.[c,h].
  - ****.cproj:*** AtmelStudio project files for each of the supported sensors.
- ***EMD-Core:*** Contains TDK driver files. These files are built into an archive libEMD-Core-ICM*.a. Each supported sensor has it's own .a file.
  - ***config :*** The Makefiles used to create the sensor driver archives.
  - ***sources/Invn:*** TDK libraries source files.
  - ****.cproj:*** AtmelStudio project files for each of the supported sensors.
- ***scripts* –** Batch files for building and flashing release versions of the firmware for each sensor.
- ***tools* –** The files required to run the host application sensor-cli.
- **EMD-G55-ICM*.atsln** – Atmel Studio solution files for each of the supported sensors.
- ***release*** – contains precompiled elf and binary files

4 *BUILDING AND RUNNING SAMPLES APPLICATIONS*

4.1 OVERVIEW

The following two projects are available:

- ***EMD-App* –** This application project demonstrates how to use TDK-InvenSense's low-level drivers to control and retrieve data from ICM devices. It encodes sensor events and sends them over the UART interface to be displayed by *sensor-cli*. The application uses the Core library and Algo libraries to generate a loadable binary.

- ***EMD-Core* –** This project includes low-level drivers and firmware code and generates the eMD Core library used by the *EMD-APP*.

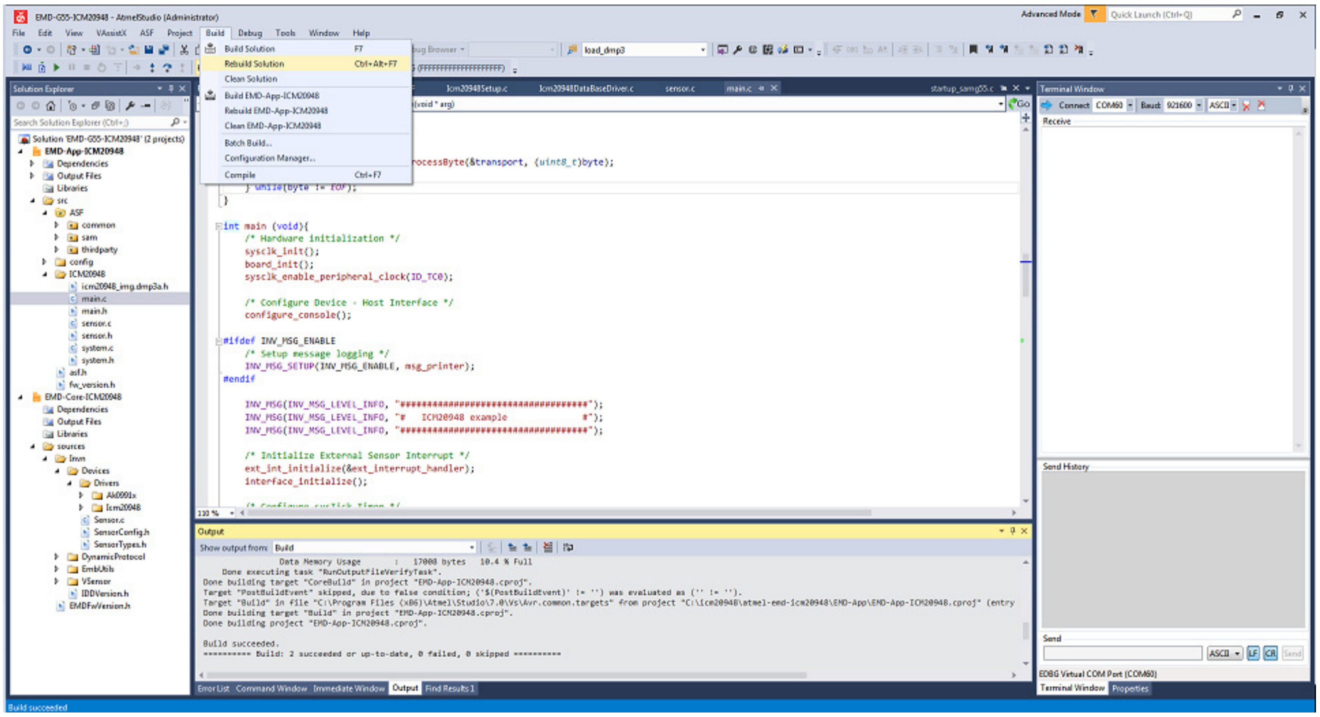
4.2 BUILDING EXAMPLE APPLICATIONS

A ready to use Atmel Studio project (EMD-G55-ICM20948.atsln) is available in the root directory.

Refer to Atmel Studio website for details on how to install Atmel Studio, building the FW and loading it to the SAMG55 Dev Kit.

Additional information is available on this document Appendix section.

Atmel Studio can be used to compile both the EMD-APP and EMD-CORE projects.



```
rocessByte(&transport, (uint8_t)byte);
    } while(byte != EOF);
}

int main (void){
    /* Hardware initialization */
    sysclk_init();
    board_init();
    sysclk_enable_peripheral_clock(ID_TC0);

    /* Configure Device - Host Interface */
    configure_console();

#ifdef INV_MSG_ENABLE
    /* Setup message logging */
    INV_MSG_SETUP(INV_MSG_ENABLE, msg_printer);
#endif

    INV_MSG(INV_MSG_LEVEL_INFO, "###################################");
    INV_MSG(INV_MSG_LEVEL_INFO, "#   ICM20948 example              #");
    INV_MSG(INV_MSG_LEVEL_INFO, "###################################");

    /* Initialize External Sensor Interrupt */
    ext_int_initialize(&ext_interrupt_handler);
    interface_initialize();
```

```
                Data Memory Usage       :   17008 bytes   10.4 % Full
        Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "EMD-App-ICM20948.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "C:\icm20948\atmel-emd-icm20948\EMD-App\EMD-App-ICM20948.cproj" (entry
Done building target "Build" in project "EMD-App-ICM20948.cproj".
Done building project "EMD-App-ICM20948.cproj".

Build succeeded.
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Figure 5 - Building application

4.2.1 Running IDD-ICM20948 example application

This application targets compatibility with low performances microcontroller (Cortex-M0, M3, …). The application instantiates directly the ICM driver and communicates through low-level APIs. The algorithms are called in the application at the frequency specified. Data is reported through the UART interface.

Atmel Studio can be used to download the compiled binary to the board via Embedded Debugger “EDBG USB” port.

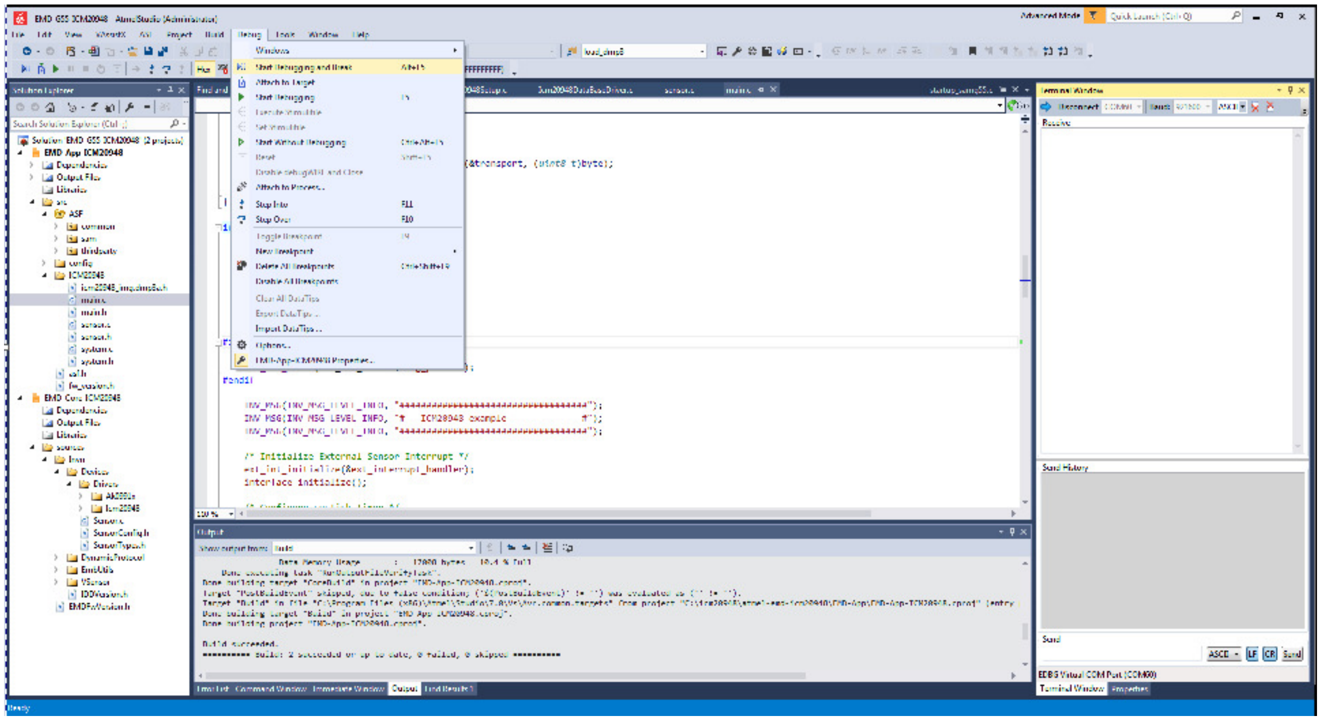


Figure 6 - Image downloading, debugging and running application

The example supports a large set of features without using a sensor framework. Some simplifications have been made to streamline the code:

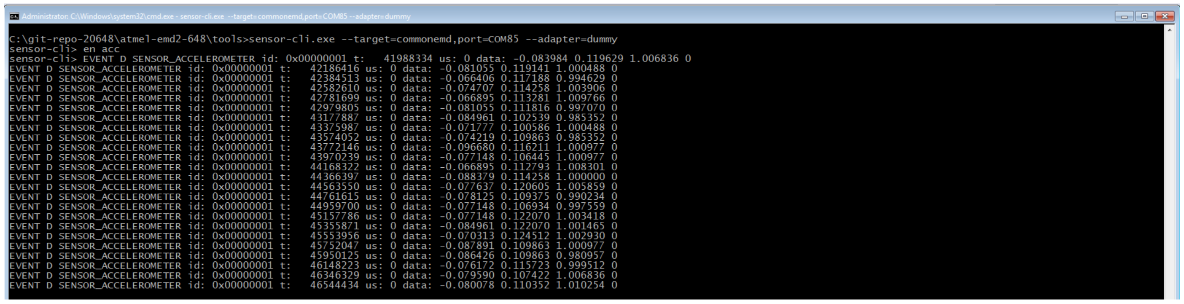
- At initialization, all algorithms are initialized and executed continuously at the default rate (50Hz)
- When an *enable_sensor* command is received, the data report is enabled (through UART)
- When a *disable_sensor* command is received, the data report is disabled but algorithms continue to run

To display data, run ***sensor-cli*** on the Windows PC from a console with the following arguments.

If only one SAMG55 system is connected to the PC, sensor-cli does not require any command line arguments. If there are multiple SAMG55 systems attached, the command should look like the following:

```
sensor-cli --target=commonemd,port=\\.\COMXX
```

Where COMXX is the comm port associated with the SAMG55 to be run.



```
Administrator: C:\Windows\system32\cmd.exe - sensor-cli.exe  --target=commonemd,port=COM85 --adapter=dummy

C:\git-repo-20648\atmel-emd2-648\tools>sensor-cli.exe --target=commonemd,port=COM85 --adapter=dummy
sensor-cli> en acc
sensor-cli> EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   41988334 us: 0 data: -0.083984 0.119629 1.006836 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   42186416 us: 0 data: -0.081055 0.119141 1.000488 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   42384513 us: 0 data: -0.066406 0.117188 0.994629 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   42582610 us: 0 data: -0.074707 0.114258 1.003906 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   42781699 us: 0 data: -0.066895 0.113281 1.009766 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   42979805 us: 0 data: -0.081055 0.111816 0.997070 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   43177887 us: 0 data: -0.084961 0.102539 0.985352 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   43375987 us: 0 data: -0.071777 0.100586 1.000488 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   43574052 us: 0 data: -0.074219 0.109863 0.985352 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   43772146 us: 0 data: -0.096680 0.116211 1.000977 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   43970239 us: 0 data: -0.077148 0.106445 1.000977 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   44168322 us: 0 data: -0.066895 0.112793 1.008301 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   44366397 us: 0 data: -0.088379 0.114258 1.000000 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   44563550 us: 0 data: -0.077637 0.120605 1.005859 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   44761615 us: 0 data: -0.078125 0.109375 0.990234 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   44959700 us: 0 data: -0.077148 0.106934 0.997559 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   45157786 us: 0 data: -0.077148 0.122070 1.003418 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   45355871 us: 0 data: -0.084961 0.122070 1.001465 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   45553956 us: 0 data: -0.070313 0.124512 1.002930 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   45752047 us: 0 data: -0.087891 0.109863 1.000977 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   45950125 us: 0 data: -0.086426 0.109863 0.980957 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   46148223 us: 0 data: -0.076172 0.115723 0.999512 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   46346329 us: 0 data: -0.079590 0.107422 1.006836 0
EVENT D SENSOR_ACCELEROMETER id: 0x00000001 t:   46544434 us: 0 data: -0.080078 0.110352 1.010254 0
```

Figure 7 – Sensor-cli console screen-shot

4.2.2 Default application behavior

At initialization, the application will:

- Initialize Atmel SAM G55 peripherals (IRQ, TIMER, SPI/I2C)
- Configure the UART (Baud-rate 2M)
- Initialize the drivers for the selected ICM device
- Setup and initialize the ICM device

4.2.3 Choosing between SPI and I2C

By default, I2C is used to communicate between ATSAMG55 and ICM device. This can be changed to SPI by setting #***define*** `USE_SPI_NOT_I2C` define to **1** (can be found in *system.h*) and by removing the jumpers between pins 1 & 2 and pins 3 and 4 of J2 (as described above).

Note: 20x48 SPI slave interface speed should not be set higher than 2.5MHz to ensure sensor data consistency

4.2.4 Configuring the device

Full Scale Range (FSR) can be changed by updating the value of the corresponding variables in *sensor.c*.

Default FSR value are **+/- 4g** for accelerometer and **+/- 2000dps** for gyroscope.

Supported FSR values are:

- Gyroscope:
  - The variable to modify is: `cfg_gyr_fsr.`
  - 250dps, 500dps, 1000dps and 2000dps
- Accelerometer:
  - The variable to modify is: `cfg_acc_fsr.`
  - 2g, 4g, 8g and 16g

Note: Accelerometer FSR is expressed in **mg** in the driver stack and application.

The array cfg_mounting_matrix (for acc and gyro) is defined in *sensor.c*. Modifying the elements of the arrays will reconfigure the mounting matrix for the associated sensors.

Default mounting matrix is set to identity which corresponds to the following reference frame:

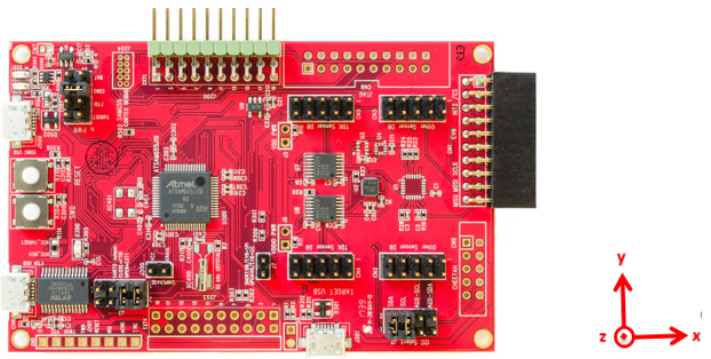


Figure 8 – board reference frame

4.2.5 Supported sensor features

- Raw accelerometer
- Raw Gyroscope
- Calibrated accelerometer
- Calibrated gyroscope
- Uncalibrated gyroscope
- Game rotation vector
- Gravity
- Linear Acceleration

Optional sensor features supported:

- Calibrated magnetometer (Inbuilt AKM9916 only)
- Uncalibrated magnetometer
- Rotation vector
- Geomagnetic rotation vector
- Step Detector
- Step Counter
- Tilt Detector
- Pick-up Gesture

- BAC (Activity Classifier)
- B2S
- SMD

4.2.6 Supported command-set

- Ping a sensor to check if it is supported by the device
- Enable / Disable sensor
- Set sensor period
- Self-tests

All sensors (except for the magnetometer) run at the same rate in the application.

All sensors are turned on at start but none is reporting yet, however data are already signaled by the ICM and processed by algorithms.

Note: Per design at start-up, all sensors and algorithms are started. The GRV orientation may drift until the gyroscope is calibrated. Once calibrated, the position is kept as the initial reference. The user may use this orientation as reference and only use the relative changes. To do so, you can refer to the following formula/code:

$$\boldsymbol{quat_{out} = quat_{grv} \,.\, conjugate(quat_{ref})}$$

With:

- $quat_{out}$ the result quaternion
- $quat_{grv}$ the quaternion obtained with the GRV sensor
- $quat_{ref}$, the quaternion that represents the position of reference

```
static void applyReferenceQuat(const float qin[4], const float q0[4], float qout[4])
{
          float q0c[4];
          // Conjugate
          q0c[0] = q0[0];
          q0c[1] = -q0[1];
          q0c[2] = -q0[2];
          q0c[3] = -q0[3];
          // Apply Compensation
          qout[0] = q0c[0]*qin[0] - q0c[1]*qin[1] - q0c[2]*qin[2] - q0c[3]*qin[3];
          qout[1] = q0c[0]*qin[1] + q0c[1]*qin[0] + q0c[2]*qin[3] - q0c[3]*qin[2];
          qout[2] = q0c[0]*qin[2] + q0c[2]*qin[0] + q0c[3]*qin[1] - q0c[1]*qin[3];
          qout[3] = q0c[0]*qin[3] + q0c[3]*qin[0] + q0c[1]*qin[2] - q0c[2]*qin[1];
          // Normalize
          float tmp = sqrtf(qout[0]*qout[0] + qout[1]*qout[1] + qout[2]*qout[2] + qout[3]*qout[3]);
          if (tmp > 0 ) {
                    qout[0] /= tmp;
                    qout[1] /= tmp;
                    qout[2] /= tmp;
                    qout[3] /= tmp;
          }
}
```

4.2.7 Frequencies

The table below sums up the achievable frequency for each sensor.

Sensor	Reporting Frequencies (Hz)		Reporting mode	Required Accel frequency (Hz)	Required Gyro frequency (Hz)
	Min	Max			
Accelerometer	1	225	*Continuous*	=	x
Raw Accelerometer	1	225	*Continuous*	=	x
Gyroscope	1	225	*Continuous*	x	=
Raw Gyroscope	1	225	*Continuous*	x	=
Uncalibrated Gyroscope	1	225	*Continuous*	x	=
Magnetometer	*1*	70	*Continuous*	x	x
Uncalibrated Magnetometer	*1*	70	*Continuous*	x	x
Game Rotation Vector	*50*	225	*Continuous*	x	x
Rotation Vector	50	225	*Continuous*	x	x

Geomag Rotation Vector	1	*225*	*Continuous*	x	x
Gravity	50	*225*	*Continuous*	x	x
Linear Acceleration	50	*225*	*Continuous*	=	x
SMD			*One shot*		
Step Counter			*On change*		
Step Detector			*On change*		
Tilt			*On change*		
Pickup			*On change*		
BAC			*On change*		
B2S			*On change*		

'=' means that the MEMS frequency will be the same as the corresponding sensor.
'x' means that it doesn't use the corresponding MEMS.

Accelerometer is raw accelerometer value scaled to output value in g.

Linear Acceleration relies on GRV and Accelerometer. When enabled the output frequency of all three sensors will be the fastest of the three.

Gravity relies on GRV. When enabled, the output of both sensors will be the fastest of the two.

Raw Gyroscope, Gyroscope and Uncalibrated Gyroscope, if enabled together, will output data at the same frequency, being the quickest one amongst f_{rgyr} , f_{gyr} and f_{ugyr}

Game Rotation Vector will output at its own frequency, no matter what other sensor frequencies are.

4.2.8 Storing self-test and algorithm bias in NV memory

Algorithm bias storage to NV memory has not been supported.

5 APPENDIX A - SYSTEM KNOWN ISSUES

- When any accelerometer-only sensor is already enabled (Accelerometer or Raw accelerometer) if a gyroscope-based sensor is enabled (Calibrated Gyroscope, Uncalibrated Gyroscope or Game Rotation Vector), system can stop reporting data for up to 50ms.

- 20x48 SPI slave interface speed should not be set higher than 2.5MHz to ensure sensor data consistency

- When Accelerometer and Gyroscope are enabled both, with different sample rate. When the accelerometer is stopped, 1 or 2 gyroscope sample are triggered with wrong rate.

- The 20x48 contains two clock division stages, as we can see in the figure below. Because the DMP engine outputs the *hw_freq* as either 1125/(1+GYRO_DIV) or 1125/(1+ACCEL_DIV) or 1125/(2^MAG_DIV) based on priorities, there are certain limitations in terms of output data rates.

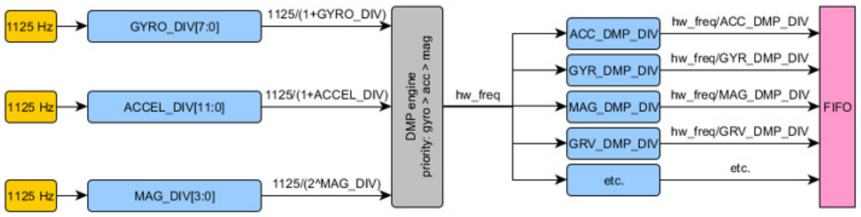
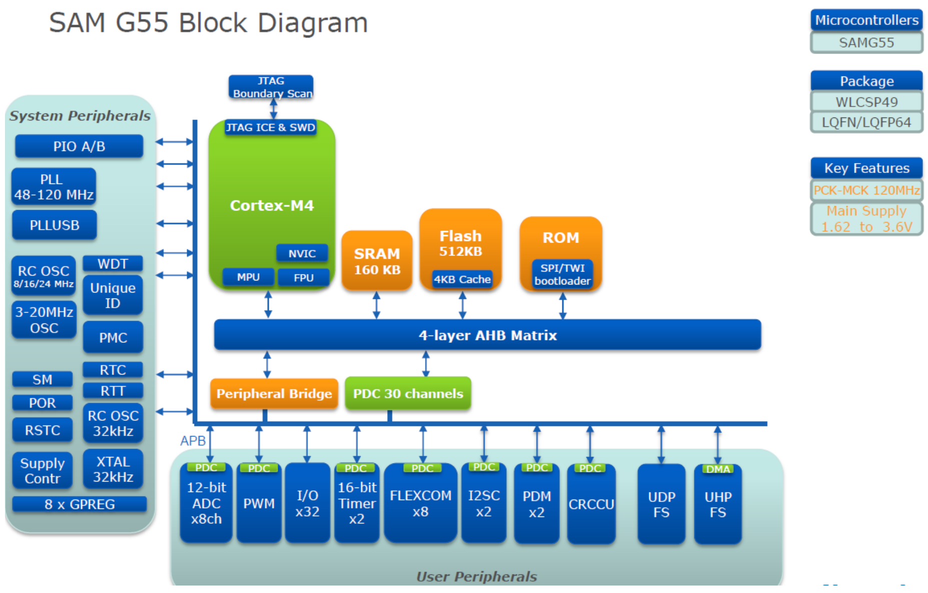
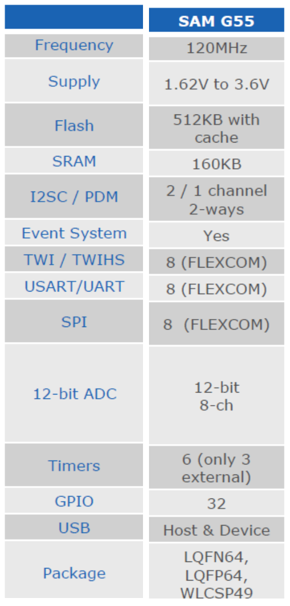


Figure 9 – Clock division stages

For example, if one requests 100 Hz ODR from the accelerometer and 50 Hz ODR from the gyroscope, the actual output data rate of the accelerometer will be 112 Hz while the actual output data rate of the gyroscope will be 56 Hz. This happens because *hw_freq* = 1125/(1+GYRO_DIV)=1125/(1+9)=112.5Hz (the gyroscope has a higher priority than the accelerometer), ACC_DMP_DIV will be set to 1 and GYR_DMP_DIV will be set to 2. As such, the actual output data rate will always be greater than or equal to the requested output data rate.

6 APPENDIX B - ATMEL SAMG55-J19 ARCHITECTURE AND SPECIFICATIONS





	SAM G55
Frequency	120MHz
Supply	1.62V to 3.6V
Flash	512KB with cache
SRAM	160KB
I2SC / PDM	2 / 1 channel 2-ways
Event System	Yes
TWI / TWIHS	8 (FLEXCOM)
USART/UART	8 (FLEXCOM)
SPI	8 (FLEXCOM)
12-bit ADC	12-bit 8-ch
Timers	6 (only 3 external)
GPIO	32
USB	Host & Device
Package	LQFN64, LQFP64, WLCSP49

Figure 9 - SAMG55 block diagram

Refer to Microchip's official site for further information on SAM-G55 architecture and system specifications.

7 DOCUMENT INFORMATION

7.1 REVISION HISTORY

REVISION	DATE	DESCRIPTION	AUTHOR
1.0	June 27, 2017	Initial version.	Rajesh Bisoi
1.1	September 1, 2017	Updated for new hardware.	Andrew Muir
1.2	September 22, 2017	Added additional configuration information	Andrew Muir
1.3	October 2, 2017	Updated included file descriptions	Andrew Muir
1.4	October 6, 2017	General cleanup	Andrew Muir
1.5	April 20, 2018	Revised.	Rajesh Bisoi
1.6	April 27, 2018	Revised.	Rajesh Bisoi

Table 1. Revision History