IMU - Invensense DK-20789

The DK-20789 is a comprehensive development platform for ICM-20789, a high performance 7-axis motion sensor that combines a 3-axis gyroscope, a 3-axis accelerometer, and a pressure sensor. The platform designed around Microchip G55 MCU can be used by developers for rapid evaluation and development of ICM-20789 based solutions.

#imu

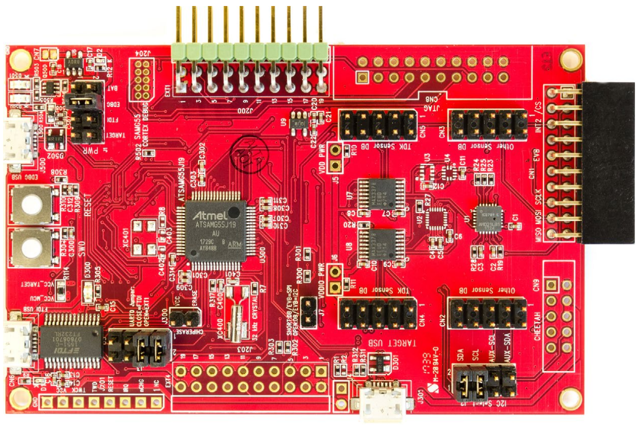
Last update: 2021-08-23 13:03:33

Table of Content

1. DK-20789

The official product homepage: https://invensense.tdk.com/products/dk-20789/

The DK-20789 is a comprehensive development platform for ICM-20789, a high performance 7-axis motion sensor that combines a 3-axis gyroscope, a 3-axis accelerometer, and a pressure sensor. This Dev Kit is compatible with the Atmel's ATSAMG55-XPRO evaluation kit which is based on a SAM G55 Cortex™-M4 processor-based microcontrollers. The supported development tools are Atmel Studio and Embedded Debugger. The ICM207xx/SAMG55 solution is an embedded sensors' combo (accelerometer, gyroscope & pressure) on chip, easy to integrate for users developing within the wearable and IoT space. The Dev Kit includes a full sensor software solution.



InvenSense DK-20789

1.1. Documents

- Firstly, check the ICM-20789 Datasheet.
- Have a quick look of different development kits which can be found in SmartMotion Platform Introduction and Training.
- The SmartMotion Hardware User Guide shows the notes and schematics of development kits.

1.2. Example projects

Register an account and download Embedded Motion Drivers (eMD) from the Download center.

There are 2 versions:

- Without Digital Motion Processor (DMP), named `eMD-SmartMotion_ICM207xx` :

  Motion processing algorithms will be run on the host processor. The host process reads all raw data and the process them. The prebuilt TDK algorithm and math libraries is not open source. It is compiled and provided as library files `libAlgoInvn.a` and `libMLMath.a` .

- With Digital Motion Processor (DMP) enabled, named `eMD-SmartMotion-ICM20789-20689-DMP`:

  Motion processing algorithms will be run on sensor itself. The host process reads all processed data when DMP sends an interrupt status. The firmware of the DMP processor is also prebuilt. It's stored in the binary array defined in `icm20789_img.dmp3.h`. This array will be uploaded to the DMP Processor when the main application runs.

The projects which are used on the DK-20789 board are built with Atmel Studio (newly changed to Microchip Studio). Download the Atmel Studio at the Microchip download page for AVR and SAM devices.

After download the eMD SmartMotion ICM-20789 DMP project, open the solution file `EMD-G55-ICM207*.atsln` to start the project.

1.3. Understanding the example projects

The example projects come with some components of InvenSense, such as Dynamic Protocol Adapter (with Data and Transport layers), and prebuilt algorithms. I haven't found any document about InvenSense's Dynamic Protocol.

The application initializes all the components, and then finally does a loop to:

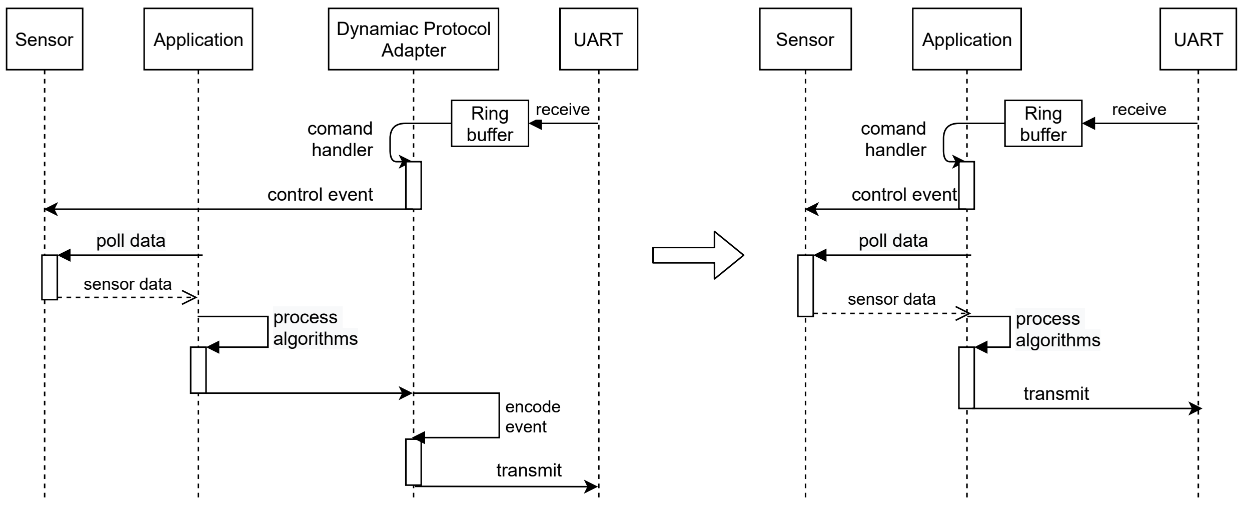
- read bytes from UART to process commands in Dynamic Protocol Adapter
- poll sensor's data when sensor sends an interrupt
- call the algorithms to process sensor's data
- converted processed sensor's data to messages for Dynamic Protocol Adapter to send through UART

⚠ Dynamic Protocol Adapter is working well with a provided example host's application named `sensor_cli`. But this program is close-source, and no document of Dynamic Protocol Adapter is found, Dynamic Protocol Adapter layer should be removed in customized projects.

Let's quickly review the application code.

1. UART ports are set up in the function `configure_console()`. The debug port is through the `FLEXCOM7` peripheral, at 921600 bps. The main console is through the `FLEXCOM0` peripheral at 2000000 bps. The console UART also enables interruption for receiving ready `US_IER_RXRDY`.
2. Sensors are set up in the function `icm207xx_sensor_setup()` and the initial configurations are set in `icm207xx_sensor_configuration()` function. By default, the Accelerator sensor at ± 4000 mg, the Gyroscope sensor at ± 2000 dps, and the Temperature sensore are enabled, and the output rate is 50 Hz.

3. The application attaches the console UART to the Dynamic Protocol Adapter with `DynProTransportUart_init()` and `DynProtocol_init()` which set callbacks to handle data in and out through the console UART port.
4. The algorithms are then initialized and configured by the `algorithms_init()` and `algorithms_configure_odr()` . Note the output data rate of the sensor should be matched with algorithm's.
5. At the boot time, all sensor output types are turned on. The variable `enabled_sensor_mask` is used as the flags to set which output types are enabled. Here is the list of sensor output types:
   - Raw Accelerator data
   - Raw Gyroscope data
   - Calibrated Accelerator data
   - Calibrated Gyroscope data
   - Uncalibrated Gyroscope data
   - Game Rotation Vector
6. Two tasks `commandHandlerTask` and `blinkerLedTask` are initialized, and started in the InvenSense's scheduler (no document about it, though its code, this not an RTOS, because it schedules tasks and run the selected task in the main loop).
7. In the main loop:
   - the scheduler will check the task which will be executed and runs it
   - call to `Icm207xx_data_poll()` function to read sensor's data when the interrupt flag `irq_from_device` is set
   - call to `sensor_event()` to forward sensor's data to the Dynamic Protocol to encode the message and transmit it to the host



The call sequence and data-flow should be modified

2. Modify the project

The example project needs to be modified to adapt to new system. The final result is to get IMU data at 200 Hz.

1. Remove Dynamic Protocol module

   As it does not have official document, this module is hard to implement in the host side. When this module is removed, the board will directly communicate with the host through UART port.

2. Remove InvenSense scheduler

   The scheduler is not needed anymore. The board will continuously send out IMU data. Removing this to save resource for printing data.

3. Change the output rate in the file `algo_eapi.h`:

   ```
   #define DEFAULT_ODR_US   5000 // 200 Hz
   ```

4. Redirect the processed sensor data

   The console UART is set to 921600 bps in the file `conf_uart_serial.h`:

   ```
   #define CONF_UART_BAUDRATE     (921600UL)
   ```

   In the function `sensor_event()` implemented in the file `sensor.c`, do not transfer data to Dynamic Protocol, add a function to call to print event data:

   ```
   void sensor_event(const inv_sensor_event_t * event, void * arg) {
       (void)arg;
       event_printer(event);
   }
   ```

5. Enable output types

   ```
   // At boot time, all sensors are turned on.
   algorithms_sensor_control(1);
   sensor_control(1);

   // Enable types of output
   enabled_sensor_mask |= (1 << SENSOR_ACC);
   ```

6. Print out sensor data

   In the main file `main.c`, implement functions to print out sensor data:

   ```
   static void console_printer(const char * str, va_list ap) {
       static char out_str[256]; /* static to limit stack usage */
       const char * ptr = out_str;
   ```

```
    vsnprintf(&out_str[0], sizeof(out_str), str, ap);
    while(*ptr != '\0') {
        usart_serial_putchar(CONSOLE_UART, *ptr);
        ++ptr;
    }
}

static void data_printer(const char * str, ...) {
    va_list ap;
    va_start(ap, str);
    console_printer(str, ap);
    va_end(ap);
}

void event_printer(const inv_sensor_event_t * event) {
    if (event->sensor == INV_SENSOR_TYPE_RAW_ACCELEROMETER) {
        data_printer("ACC_RAW,%d,%d,%d\r\n",
            event->data.raw3d.vect[0],
            event->data.raw3d.vect[1],
            event->data.raw3d.vect[2]
            );
    }
}
```

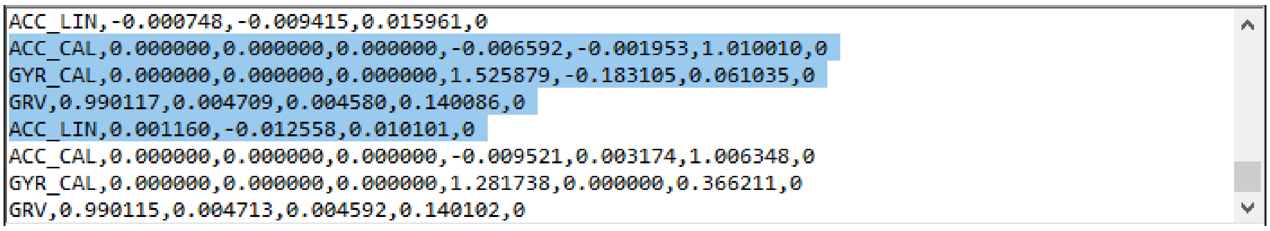
Repo of the modified project

The modifications are tracked in a repo at: https://github.com/vuquangtrong/eMD-SmartMotion_ICM207xx.

The output

Here is an example of outputting 4 data types from sensor:

- Calibrated Accelerator
- Calibrated Gyroscope
- Game Rotation Vector
- Linear Acceleration



```
ACC_LIN,-0.000748,-0.009415,0.015961,0
ACC_CAL,0.000000,0.000000,0.000000,-0.006592,-0.001953,1.010010,0
GYR_CAL,0.000000,0.000000,0.000000,1.525879,-0.183105,0.061035,0
GRV,0.990117,0.004709,0.004580,0.140086,0
ACC_LIN,0.001160,-0.012558,0.010101,0
ACC_CAL,0.000000,0.000000,0.000000,-0.009521,0.003174,1.006348,0
GYR_CAL,0.000000,0.000000,0.000000,1.281738,0.000000,0.366211,0
GRV,0.990115,0.004713,0.004592,0.140102,0
```

Data types output from ICM-20789 sensor

3. Appendix

Digital Motion Processor

The embedded Digital Motion Processor (DMP) offloads computation of motion processing algorithms from the host processor. The DMP acquires data from the accelerometer and gyroscope, processes the data, and the results can be read from the FIFO.

The DMP has access to one of the external pins, which can be used for generating interrupts. The purpose of the DMP is to offload both timing requirements and processing power from the host processor.

Typically, motion processing algorithms should be run at a high rate, often around 200 Hz to provide accurate results with low latency. This is required even if the application updates at a much lower rate; for example, a low power user interface may update as slowly as 5 Hz, but the motion processing should still run at 200 Hz.

The DMP can be used to minimize power, simplify timing, simplify the software architecture, and save valuable MIPS on the host processor for use in applications. DMP operation is possible in low-power gyroscope and low-power accelerometer modes.