DRONE SITL BRINGUP WITH THE IIO FRAMEWORK

Bandan Das <bsd@redhat.com>

Open Source Summit, Europe, 2018
WHAT'S THIS ABOUT?

- My experiments with bringing up sensors on a x86 board
- Understanding the IIO framework
  - Interfacing the framework with SITL code to verify sensors are working
WHAT ISN'T THIS ABOUT?

- Flying
- Drones, I am a newbie!
ACKNOWLEDGEMENTS

- Real Time systems group at BU
- https://www.cs.bu.edu/~richwest/index2.html
- Microkernels, Cache scheduling algorithms, Virtualization, Predictable time
STATE OF THE ART
Most drone boards belong to the STM32 family
Becoming faster/powerful everyday!
Low power requirements
SOFTWARE

• Ardupilot, Betaflight, iNav, Cleanflight
WHY X86?

- Increasingly complex tasks and onboard peripherals
- Processing power
- The case for reactive drones
- Low power x86 boards are a reality although not common
WHY LINUX?

- Robust operating system
- Drivers for a wide variety of sensors/peripherals
- Choice of schedulers
A LOOK AT THE INTEL AERO COMPUTE BOARD

Aero as a Companion Computer
HOWEVER...

- The Aero is a powerful computer
- Onboard sensors to run standalone as a flight controller
- Access to GPIO pins/motor outputs through onboard FPGA
MOVING FC OPERATIONS TO A X86 BOARD

- Reinventing the wheel
  - Customized Linux that jumps to a flight controller loop with specific tasks
- Leverage on existing solutions
  - Run an existing flight controller software as a process
INTERFACING WITH THE ONBOARD SENSORS

- `spidev/i2cdev` - userspace drivers
- Linux already has drivers for most sensors
  - exposed by the Industrial IO interface (IIO)
- Advantages
  - Minimizes latency
- Disadvantages
  - Crashes can be deadly
AN INTRODUCTION TO THE IIO INTERFACE

- Industrial Input/Output
  - Examples include Humidity Sensors, Temperature sensors, Magnetometer etc
  - v4.18: ~20 classes, each containing numerous device drivers
  - Most devices connected via I2C or SPI
- IIO provides a hardware abstraction layer over these devices
  - Sharing of infrastructure
  - Developer focus on device function rather than knowledge of plumbing internals
  - Consistent application development framework
  - Data buffer for continuous data and single shot access via sysfs
GETTING STARTED WITH IIO

- Device drivers
  - BMI 160 Inertial Measurement Unit
    ```
    ls drivers/iio/imu/bmi160/
    bmi160_core.c bmi160_i2c.c bmi160_spi.c
    ```
  - BMM 150 3 axis Geomagnetic Sensor
    ```
    ls drivers/iio/magnetometer/
    bmc150_magn.c bmc150_magn_i2c.c bmc150_magn_spi.c
    ```
  - MS5611 Pressure Sensor
    ```
    ls drivers/iio/pressure/
    ms5611_core.c ms5611_spi.c ms5611_i2c.c
    ```
GETTING STARTED WITH IIO

- Key components
  - Device drivers
  - Channels
  - Buffers
  - Triggers
GETTING STARTED WITH IIO

- Channels - One of the many functions provided by the device
  
  - **BMI160**
    
    cat /sys/bus/iio/devices/iio\devic0\name
    bmi160
    ls /sys/bus/iio/devices/iio\devic0/
    in_accel_x_raw in_accel_y_raw in_accel_z_raw
  
  - **BMM150**
    
    cat /sys/bus/iio/devices/iio\devic1\name
    bmc150\magn
    ls /sys/bus/iio/devices/iio\devic1
    in_magn_x_raw in_magn_y_raw in_magn_z_raw
GETTING STARTED WITH IIO

- Buffers
  - Raw continuous data read from the device
  - Specific channels can be enabled
  - Data format specified by channels
    - Example:
      ```
      cat /sys/bus/iio/devices/iio\:device1/scan_elements/in_magn_x_type
      le:s32/32>>0
      ```
  - Kfifo backed
  - Read using standard fileops by accessing /dev/iio:deviceX
  - mmap based interface supported by a DMA backend (high speed devices)
GETTING STARTED WITH IIO

- Triggers
  - Capture data only when needed
    - Based on a hardware event
    - User initiated (eg: via sysfs)
    - Software trigger (eg: hrtimer based)
    - Enabling trigger enables data capture
GETTING STARTED WITH IIO

- Initializing SPI/I2C devices
  - Not enumerated at the hardware level
  - SPI (BMI160)
    - Device tree
    - ACPI
    - Board initialization file
  - Example:

```c
static struct spi_board_info imu_board_info __initdata = {
    .modalias = "bmi160",
    .bus_num = SPIDEV_SPI_BUS,
    .chip_select = SPIDEV_SPI_CS,
    .max_speed_hz = SPIDEV_SPI_HZ,
};
...
master = spi_busnum_to_master(SPIDEV_SPI_BUS);
...
dev = spi_new_device(master, &imu_board_info);
...```


GETTING STARTED WITH IIO

- Initializing SPI/I2C devices
  - I2C (BMM150)
    - Device tree
    - Board initialization file
    - sysfs interface
    - Example:
      ```
      echo bmc150_magn 0x12 > /sys/bus/i2c/devices/i2c-2/new_device
      ```
CREATING A TRIGGER

mkdir /sys/kernel/config/iio/triggers/hrtimer/trigger0
echo 5000 > /sys/bus/iio/devices/trigger0/sampling_frequency
cd /sys/bus/iio/devices/iio:device0 #Associate trigger with BMI160
echo trigger0 > trigger/current_trigger
echo 1 > scan_elements/in_accel_x_raw
echo 1 > buffer/enable
SHIM LAYER: LIBIIO

- Library that interfaces with the IIO API
- Ease of developer interacting with the IIO framework
SIMULATION FRAMEWORKS (SITL/HITL)

- **SITL**
  - Modified flight controller software running in a simulator environment
  - Control signals come from software or a controller
  - Simulator feeds sensor data back to firmware feedback loop
  - Actuator outputs fed to simulator

- **HITL**
  - Flight controller software runs on the actual board
  - Sensor data and outputs fed to a simulator
  - Enables testing in closer to real-world conditions
Basic SITL setup in Betaflight/Cleanflight
typedef struct {
    double timestamp;
    double imu_angular_velocity_rpy[3];
    double imu_linear_acceleration_xyz[3];
    double imu_orientation_quat[4];
    double velocity_xyz[3];
    double position_xyz[3];
} fdm_packet;
PLUGGING IN IIO DATA IN THE SITL LOOP
PLUGGING IN IIO DATA IN THE SITL LOOP

- Visual indication of correct functioning of sensors
- Sensors output in a more readable format or graph (eg. Cleanflight Configurator)
BETAFLIGHT TASKS

*/ TASK_COUNT = ~30 */
cfTask_t cfTasks[TASK_COUNT] = {
...
  [TASK_GYROPID] = {
    .taskName = "PID",
    .subTaskName = "GYRO",
    .taskFunc = taskMainPidLoop,
    .desiredPeriod = TASK_GYROPID_DESIRED_PERIOD,
    .staticPriority = TASK_PRIORITY_REALTIME,
  },
  [TASK_ACCEL] = {
    .taskName = "ACC",
    .taskFunc = taskUpdateAccelerometer,
    .desiredPeriod = TASK_PERIOD_HZ(1000),      // 1000Hz, every 1ms
    .staticPriority = TASK_PRIORITY_MEDIUM,
  },
  [TASK_ATTITUDE] = {
    .taskName = "ATTITUDE",
    .taskFunc = imuUpdateAttitude,
    .desiredPeriod = TASK_PERIOD_HZ(100),
    .staticPriority = TASK_PRIORITY_MEDIUM,
  },
  ...
#ifdef USE_MAG
  [TASK_COMPASS] = {
    .taskName = "COMPASS",
    .taskFunc = compassUpdate,
    .desiredPeriod = TASK_PERIOD_HZ(10),        // Compass is updated at 10 Hz
    .staticPriority = TASK_PRIORITY_LOW,
  },
#endif
...
IMU LOOP COUNT

- Standard upstream kernel
- No specialized config
IMU LOOP COUNT

- Standard upstream kernel
- isolcpus=2-3
- irq thread on CPU 2 and read process on CPU 3
REFERENCES

- https://bitbucket.org/bdas/iio_sensors (Code snippets for the SITL/IIO interface)
- https://lwn.net/Articles/370423/ (Secrets of the Ftrace function tracer)
- https://github.com/betaflight/betaflight (Betaflight source)
- https://github.com/analogdevicesinc/libiio (Library for interfacing with IIO devices)
- https://www.youtube.com/watch?v=ealH3qP_pBE (APM on Linux: Porting Ardupilot to Linux1)
- https://archive.fosdem.org/2015/schedule/event/iiosdr/ (Using the Linux IIO framework for SDR)
- https://www.cs.bu.edu/~richwest/index2.html (Rich West's Home page)
WRAP UP NOTES

• Running a flight controller software as a process
• Interfacing with IIO appears to be straightforward
• Further investigation on latency and performance
• Running a PREEMPT_RT kernel
• More experiments with affinities