Research Activity and Practical Work

I2C slave driver for reading multiple IIO channels

Sergiu-Cristian Ionescu

Politehnica University of Bucharest
Faculty of Electronics Tellecomunications and Information Technology
Advanced Computing in Embedded Systems

1 Introduction

Today, one of the most common applications in embedded systems is data acquisition. Usually the architecture of such systems consists of a development board and some sensors. The first important part to consider is what board should be used and how can a user interact with it. A FPGA development board is often used for data acquisition applications as it is fast and very flexible in its configurations. However as I said we also must think of user interactions and practical applications for this system. That's why it is usually preferred to have an operating system on the board, in order to have user space applications that can interact with the hardware, process the input data and also make it available on a network in a much convenient way. Linux is the often the choice in embedded applications as it is open source and very efficient in its implementation. Linux has various subsystems that permits the user-space programs to interact with the hardware (ex : i/o subsystem). However, for data acquisition a better subsystem was needed, one that can interact with faster rates sensors and be more precise in its processing. Thus the IIO subsystem appeared which is used in this type of acquisition systems. Another important thing to consider it is how the sensors would communicate with the development board ? Usually a serial communication protocol like I2C or SPI is used for sensors.

In the future I will to develop an I2C slave driver that can read multiple IIO channels and for a development board, I am going to use a PYNQ-Z1 board. In order to prepare, I chose for this research activity to find more about the IIO and I2C kernel subsystems and the way they are used in data acquisition systems.

2 Research keywords

- I2C
- IIO
- Subsystem
- Sensors
- Data acquisition
- Driver
- FPGA

3 Selected Papers:

- Rauby, Pierrick. Developing a smart and low cost device for machining vibration analysis. Diss. Georgia Institute of Technology, 2018.
- Simha, AP Vikram. A real-time embedded data acquisition system for surface measurements using multiple line lasers. The University of Texas at Arlington, 2015.
- Téllez Chica, Agustín. ”Driver Linux con soporte Industrial I/O para sensor de consumo.” (2020). (Industrial I/O Linux driver for power monitor)
- Shao, Hong, et al. ”The Design of a Hospital Environment Data Acquisition System Based on ARM11 and Embedded Linux.” Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering. Atlantis Press. 2013.
- Karcher, Nick, et al. ”Versatile Configuration and Control Framework for Real Time Data Acquisition Systems.” IEEE Transactions on Nuclear Science (2021).

4 Rejected Papers:

- Allwright, Michael, Weixu Zhu, and Marco Dorigo. ”An open-source multi-robot construction system.” HardwareX 5 (2019): e00050
- Suciu, Adrian. ”Using GNU Radio to do signal acquisition and analysis with Scopy.” (2018).
- Potter, David. ”Using Ethernet for industrial I/O and data acquisition.” IMTC/99. Proceedings of the 16th IEEE Instrumentation and Measurement Technology Conference (Cat. No. 99CH36309). Vol. 3. IEEE, 1999.
- Zheng-xiang, ZHU Nan-hao LI. ”Research and implementation of I2C device driver under embedded Linux system.” 26 (2010): 4-2.
- Kalomiros, John, John Vourvoulakis, and Stavros Vologiannidis. ”A Workflow for Designing Video Processing Pipelines with PYNQ.” 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). Vol. 1. IEEE, 2021.
- Arif, Rizki, et al. ”Development of electroencephalography (EEG) data acquisition system based on FPGA PYNQ.” AIP Conference Proceedings. Vol. 2092. No. 1. AIP Publishing LLC, 2019.

5 IIO and I2C subsystem introduction

5.1 IIO subsystem

The IIO subsystem was mainily created to read data from anything that is at heart an Analog to Digital Converter or Digital to Analog Converters, which are often used for sensors. It allows a user-space interface via the SYSFS and uses Chrdev based FIFOs and Events to control it. It also uses triggers to sample the data from the enabled channels

As you can see from the picture above, the IIO subsystem interacts with the User Space using the SYSFS. After that we can see the main components of the subsystem : The IIO core, the IIO device and the IIO Bus Driver. These will be further detailed with their usage in the following research papers.
The main features of this subsystem are the following :

- IIO channels
- Triggers (SW and HW based)
- Data Buffers
- Single-shot Data Access
- IIO Events
- IIO Channel Consumers

This subsystem is usually used when we use high speed sensors and we need a per sample timestamp which would be important for fusion code integration.
Usually when we want to implement such a driver we must first configure the iio$_{c}han_{s}pecstructure.Thisisthemostimportan$
$Furtherdetailsfortheirimplementationswillbediscussedinthefollowingpapers.$

5.2 Protocol basics

I2C is a serial protocol widely used in embedded applications due to its easy usage. It's a slow protocol but easy to configure. It basically consists from one or multiple masters and its slaves. Only the masters can control the slaves.
Each slave has its unique address. Transfer from and to the master device is serial and it is split into 8-bit packets. The protocol uses 2 wires:

- SCL (serial clock)
- SDA (serial data) – used by the Controller Transmitter and by the Controller Receiver

Each I2C slave device has a 7-bit address that needs to be unique to the bus. The 7-bit address represents bits 7 to 1 while bit 0 is used to signal reading from or writing to the device. If bit 0 in the address byte is set to 1 then the master device will read from the slave I2C device.

The master device needs no address since it generates the clock (SCL) and addresses individual I2C slave devices.

In a normal state, both lines (SCL and SDA) are high. The communication is initiated by the master device. It generates the Start condition followed by the address of the slave device (first byte B1). If bit 0 of the address byte was set to 0 the master will write to the slave device data (second byte B2). Otherwise, the next byte will be read from the slave device. After all the bytes are written or read, the master device generates Stop Condition (P). Furthermore, sometimes we want to read data from the slave, and also to write to it In order to do that we can repeat the start condition , by sending again the slave address changing the 0 bit in the address byte from write to read or from read to write. This will happen before the Stop Condition.

Another important thing about the transfer, is that after each byte we need to receive an acknowledge bit from the Slave This bit signals whether the device is ready to proceed with the next byte. For all the data bits including the Acknowledge bit, the master must generate clock pulses. If the slave device does not acknowledge transfer this means that there is no more data or the device is not ready for the transfer yet.

For the transfer part the Controller Transmitter is used and for the receiving part (receive acknowledge bit from slave) the Controller Receiver is used. Though they are 2 different signals, they share the same physical bus (SDA).

Each master must generate its own clock signal and the data can change only when the clock is low. For successful bus arbitration, a synchronized clock is needed. Once a master pulls the clock low it stays low until all masters put the clock into a high state. Similarly, the clock is in a high state until the first master pulls it low. This way by observing the SCL signal, master devices can synchronize their clocks.

Microcontrollers that have dedicated I2C hardware can easily detect bus changes and behave also as I2C slave devices. However, if the I2C communication is implemented in software, the bus signals must be sampled at least two times per clock cycle in order to detect necessary changes.

5.3 I2C kernel subsystem

This subsystem is divided in Buses and Devices, which are managed by the I2C core. The Buses are divided into Algorithms ad Adapters, and the Devices are divided into Drivers and Clients.

The Algorithms contain code that can be used for a whole class of I2C adapters.

The Adapters are used to tie up a particular I2C with an algorithm and bus number. Each specific adapter driver either depends on one algorithm driver or includes its own implementation.

The Clients represent the slave physical chips on the I2C.

The Drivers are those that assure the communication from user space to the physical space with the slaves, using the respective protocol. The protocol is however specifically implemented in the Algorithm part.

Clients and Drivers

These two are usually closely integrated.

In order to create such a driver we must do the following steps:

- Get the I2C adapter
- Add the new device
- Create the driver
- Transfer the Data
- Unregister the device and delete the driver

In order to get the adapter we must define the following structure :
struct i2c_adapter * i2c_get_adapter(int nr);

In order to create a device we must first define the following structure:
struct i2c_board_info();

After we got the adapter and the board info we instantiate the device using the **i2c_new_device** function, which takes as parameters a **i2c_adapter** structure, and a **i2c_board_info** structure.

It's important to define the device, because this represents the slave. When we want to send some data to the slave we would use functions like **i2c_master_send** which takes as parameter a device structure. This is because in order to communicate with the slave on the bus we use the Bus part of the subsystem, which needs to know what adapter to use for the slave device.

After we define the slave, we must add our driver to the subsystem. The driver is also defined by a structure named **i2c_driver**. This structure contains the probe, remove and id functions which are usually used in drivers. So we must first define our probe, remove and id. When we write our probe function for example, if we want to send something initially to the slave, we'll again use a function like **i2c_master_send** which needs an **i2c_device** as a parameter. After we define this structure, we can add it to the subsystem using the **i2c_add_driver** function

6 Developing a smart and low cost device for machine vibration analysis

In this paper the author uses a BeagleBone Black development board in order to monitor vibrations on a bandsaw and determine the nature of the material that is being cut by the bandsaw. The data is registered using specific sensors and it is retrieved and sampled by the board using the Industrial I/O subsystem, as the board is running Linux on it. After the data was acquisitioned by the kernel subsystem it is later classified using machine learning. The part of interest for me, is the data acquisition as it is using IIO and I am interested in how the setup was made, and the way the board was configured.

The chip that runs on the BeagleBone board is the ti-am3358. This chip has 2 Proccess Realtime Units and one ARM Cortex. The author describes the main functionalities of the Linux Operating System and then it starts to describe the Industrial I/O subsystem. This subsystem is being used as usual to get data from ADCs to the board. This subsystem is especially preferred because it allows interrupts to be used, which are suitable for deterministic and data sampling at high rates. Another important fact that is being described it is that the drivers can be easily loaded and unloaded to tell the kernel how to deal with the particular hardware. Thus the Linux operating can show his power in its versatility.

Here in this project, the author uses the 4 main parts of the subsystem : The **iio_ring**, the **iio_core**, the **iio_trigger** and the **iio_device_driver**. Also the author uses a device tree in order to set a hardware description of the ADC and the clock frequency of the board. For the entire IIO subsystem (IIO driver) the author uses an already written device driver by Texas Instruments for the specific sensor. This is described briefly as receiving and transmitting information to the user space through the sysfs interface.

As stated next in this project, after we have the data acquisitioned and ready to be read from the character device, we need a method to read correctly this data. Thus a user space application was written in order to disable the **iio_triger** (in order to have the data stored), activate the **iio_channel**, create a buffer directory to set the buffer length, create a csv file with time_stamp and sample value and read the values from the buffer to the csv file.

So as a conclusion to this project, it is important to understand that we often use the IIO subsystem because it lets us to register data from sensors that work at very high frequencies. Another importing thing is to know how to configure the main components of our system using the device tree files, and also we can always find already made implementations of the IIO device drivers. Last but not least we need to read the data correctly from the device character after the acquisition was made by enabling the IIO_CHANNELS and disabling the IIO_TRIGGERS.

7 A real-time embedded data acquisition system for surface measurements using multiple line lasers

In this research project we have again a data acquisition system using a development board and some sensors. The main interest for us is again the use of the IIO subsystem. Here the author chose to use an Intel Galileo board. After that, in chapter 5.3.2 it describes the main utilities of this system. In addition to the previous paper, the author enhances the idea of controlling various parameters through the sysfs interface, by using the IIO subsystem : the sampling rate, number of channels, buffer size etc.

Furthermore, the author explains in more detail the IIO core, IIO trigger, IIO ring and the IIO device drivers. The important things to remember is that the IIO devices are registered to the IIO subsystem as IIO device drivers, which are connected to the hardware through standard interfaces like I2C or SPI. The IIO trigger is used to take the messages from the driver, pass them to the IIO core and IIO ring layers, and from there to the user space. For each trigger input, the device driver gets the sampled data from the driver. The data which is written into the device, is written in a ring buffer, which is used to solve the fast producer – slow consumer problem.

Furthermore, we can have various trigger available :

- Sysfs triggers for singled value operations, useful for verification
- IIO HR timer triggers, useful for continuous operations.

Also the author uses an already written IIO driver code, for the sensor which is included in the Linux kernel. So we already have the core, triggers and ringalready configured. The main thing we need to do is to correctly load the module in the kernel.

As a conclusion to this research project, we saw a further explanation of the IIO subsystem and also its usage in the data acquisition. We also saw that in order to the have the data correctly registered, we need to use continuous operations, thus activating the HR timer triggers, and configuring the IIO ring buffer. The ring buffer is also a very important part because its prevents the data to be read to fast and processed to slow. Again, we saw that the driver is already implemented, the most important thing being to know how to use it.

8 Driver Linux con soporte Industrial I/O para sensor de consume (Industrial I/O Linux driver for power monitor)

Here I studied a Spanish research project. The author retrieves data here using the IIO subsystem on the Raspberry PI, but this time it also communicates with the sensor using the I2C interface. The sensor that he is using is an INA219. First the author describes the functionalities of the sensor and the GPIO pins of the Raspberry Pi available for I2C communication. It is important to know that each sensor has a different type of channel. Some sensor measure temperature, some measure voltage etc. When we configure the IIO acquisition channels in the driver, it is important to specify the type of channels that we will use. In our case we have a voltage, current and power measuring sensor.

An IIO device normally corresponds to a unique hardware which is managed by a controller.

In the IIO subsystem there are structures which are defined in order to manage the devices :

- iio_dev : which defines the IIO device
- iio_device_alloc() and iio_device_free which assgins or free a iio_dev to a controller

- iio_device_register() and iio_device_unregeistre whih registers/unregisters a device in the IIO subsystem.

For the iio_device there are various working modes : INDIO_DIRECT_MODE, INDIO_BUFFER_TRIGGERED, INDIO_BUFFER_HARDWARE and INDIO_ALL_BUFFER_MODES.

In order to interact from user to space to an IIO controller we can either access the /sys/bus/iio/iio:device which returns all the IIO channels or the /dev/iio:device which is used to get the data from the buffer of a specific channel.

An IIO driver usually registers for controlling I2C or SPI interface, by using the probe method.

When we use the probe method a memory space is reserved for the device by using the iio_device_alloc(). Also we have the initial information configuration of the device like the number of channels that would be used.As I mentioned in the beginning it is very important to configure the IIO channels. A channel in an IIO controller represents a channel of data. A sensor can hold multiple data channels, so we need to configure the hardware properly. The sensor in this project has 3 data channels : IIO_VOLTAGE, IIO_CURRENT, and IIO_POWER. There are also other parameters, beside the type, like info_mask_separate, info_mask_shared_by_type/dir/all. The info_mask_separat attribute, represents the unique characteristics of the data channel. The info_mask_shared_type/dir/all attribute, represents the common characteristics of the data channel with the other channels of the hardware device.

After that we have the i2c driver implementation presented. Here again we see that in order to initialise the i2c driver we use the specific init function ina219_i2c_init. We also need to add the driver in the device table with the MODULE_DEVICE_TABLE table and define the driver name, prove, remove and id_table in the i2c_driver struct. We also need to configure the file operation struct and the device tree correspondingly.

9 The Design of a Hospital Environment Data Acquisition System Based on ARM11 and Embedded Linux

The next paper introduces again a similar data acquisition method. Here the data is registered and processed for a hospital environment. The author first talks about the limited computing speed, storage space and processing capability of a MCU. An operating system with a real time and multi tasking capabilities cannot be run on a MCU. So the author opts for an ARM processor usage, as they are good for monitoring systems and are becoming more and more powerful. An ARM based system can collect a variety of environment data. The ARM11 development board is connected to a data acquisitions board which had temperature, humidity and light sensors integrated in it. The ARM development board is used to deal with data processing and display, while the data acquisition board is used to deal with the data acquisition. The project has a hardware part and a software part. I will discuss the software part from the paper, as this is the part which uses the IIO subsystem to retrieve data from the sensors.

The embedded Linux includes the Uboot, Linux Kernel, yaffs2, Qt, device driver etc. The system can work normally only if the acquisition device worked with the highly efficient and flexible device.

Here the author talks about the main cores of the I2C subsystem in the Linux as well as interfacing it with the IIO subsystem.

What is important to remember is that the I2C subsystem has 3 main parts : The i2c Core, the i2c adapter, and the i2c equipment driver. The i2c adapter is a driver which is used to access the hardware/sensor. The i2c core is used to add the hardware to the i2c bus and permits all the access methods to the i2c equipment driver. And last but not least, the i2c driver is used to access the user space. Through the i2c_device we can access each of the 3 main parts through the user space.

Furthermore we need in the driver implementation to define a i2c_driver struct in order to register the I2C equipment driver to the I2C core in the initialization module, and complete the related operation of the I2C_adapter.

So as we can see in this paper, we can see the importance of a flexible and efficient acquisition software method in order to function properly with the hardware.

10 Versatile Configuration and Control Framework for Real-Time Data Acquisition Systems

Last paper I will discuss here brings in discussion the idea of framework included in an FPGA, in order to control its hardware, also for Data Acquisition. The paper implies that using a system-on-chip FPGA solution with a microprocessor include, can include more sophisticated configurations and calibrations solutions, as well as standard remote access protocols that can be efficiently integrated into the software. The software framework would implement convenient access to the hardware and a flexible abstractions via remote-procedure calls. Control flow, complex algorithms, data conversions, and processing, all of them can be provided to a client via Ethernet.

So the paper implies the importance of using the FPGA in Data Acquisition Systems as they are favourable for large-scale experiments that contain dozens of measurements over a longer period of time.

Having this in mind using the PYNQ-Z1 board for data acquisition is a good idea as it is also an FPGA with an included microprocessor.

11 Conclusions

As we can see from these papers, the IIO and I2C subsystems are widely used in embedded data acquisition systems. It is very important to understand their architecture and the way they can access the user space. Moreover the FPGA boards are preferred for faster and more flexible acquisitions. Hence, an implementation of a driver that has an I2C device read from multiple IIO channels can prove to be very useful in the future.

12 Bibliography

IIO subsystem:

- Industrial I/O and You: Nonsense Hacks! - Matt Ranostay, Konsulko Group - presentation at The Linux Foundation ; Industrial I/O and You: Nonsense Hacks! - Matt Ranostay, Konsulko Group - YouTube ; PowerPoint Presentation (elinux.org)
- 10 Years of the Industrial I/O Kernel Subsystem - Jonathan Cameron, Huawei – presentation at The Linux Foundation: 10 Years of the Industrial I/O Kernel Subsystem - Jonathan Cameron, Huawei - YouTube; ELC_2017_- _Industrial_IO_and_You-_Nonsense_Hacks!.pdf (elinux.org)
- Linux Industrial I/O Subsystem [Analog Devices Wiki]
- https://www.kernel.org/doc/html/v4.12/driver-api/iio/index.html

I2C subsystem:

- I2C Tutorial - Basic Introduction (Part 1) EmbeTronicX
- I2C Protocol Tutorial - PART 2 (Advanced Topics) EmbeTronicX

- I2C Linux Device Driver using Raspberry PI - Device Driver 37 EmbeTronicX
- I2C Bus Driver (Linux Device Driver Tutorial) EmbeTronicX