the gluion
advantages of an FPGA-based sensor interface

Sukandar Kartadinata
glui
Hagenauerstr. 6
10435 Berlin, Germany
sk@glui.de

ABSTRACT

The gluion is a sensor interface that was designed to overcome some of the limitations of more traditional designs based on microcontrollers, which only provide a small, fixed number of digital modules such as counters and serial interfaces. These are often required to handle sensors where the physical parameter cannot easily be converted into a voltage. Other sensors are packed into modules that include converters and communicate via SPI or I2C. Finallly, many designs require output capabilities beyond simple on/off.

The gluion approaches these challenges thru its FPGA-based design which allows for a large number of digital I/O modules. It also provides superior flexibility regarding their configuration, resolution, and functionality. In addition, the FPGA enables a software implementation of the host link - in the case of the gluion the OSC protocol as well as the underlying Ethernet layers.

KEYWORDS

Digital Sensors & Actuators, Sensor Interfaces, FPGA, OSC.

1. INTRODUCTION

The renewed interest in sensor technology in recent years has resulted in an enormous pool of resources, both in terms of online information and available tools. Part of this trend is an ever-growing number of sensor interfaces each with its own special feature set, concept, and price range. What they all have in common, however, is the fact that they are built around a microcontroller. This seems like a perfectly reasonable choice as these integrated circuits combine everything that is required for the design of a sensor interface: a CPU to handle program flow and perform arithmetics, memory to store programs and data, and I/O modules. The latter are of particular importance as they allow the microcontroller to connect directly to both sensors and host computer without the use of additional I/O chips. For the host link this is usually a serial interface like MIDI or RS232, or, in more recent designs, USB or Ethernet.

On the sensor side the most important I/O modules are the analog inputs that accept the voltages generated by the sensors. These are then measured by an integrated Analog-to-digital converter for subsequent processing by the CPU and transmission towards the host. In addition, microcontrollers provide a number of digital I/O pins that can be controlled, or queried, under program control. They can also be associated with additional internal modules to perform dedicated functions. A typical example is a PWM (Pulse-Width Modulation) generator that is often used to control the speed of a motor. This is usually implemented by using a timer module whose value is constantly compared to a register that contains the desired pulse-width. The (binary) result of this comparison then leaves the chip directly thru a dedicated pin which is connected to the motor circuitry.

Now, while the above approach is fine for many smaller projects, it is not uncommon that an instrument or installation design requires more than just a few PWM-, or other digital I/O-, modules. What's more is that the timers are also required for other tasks, e.g. generating the MIDI clock. Unfortunately, simply switching to a bigger microcontroller only helps to a certain extent as the focus is usually on more memory and general purpose pins rather than I/O modules. And adding more microcontrollers complicates the overall system design as they have to communicate with each other.

FPGAs, in contrast, do not have these limitations. As the name, Field Programmable Gate Array, suggests, the building blocks of these devices are gates, the logic foundation of any digital device. However, rather than wiring up countless TTL chips like in the early days, the designer can enter the schematic with a suitable editor[1] which is then compiled into a configuration file for the FPGA. This means that before programming the microcontroller one would first design it according to the requirements. While this may seem like reinventing the wheel, modern design tools hide a lot of the complicated details while the added flexibility outweighs the extra effort. More to the point though, we will see that many components of the microcontroller can be left out of the design.

2. BACKGROUND

The gluion is not the first device that employs FPGAs in a music technology context. Two examples for their use in custom controller designs are Dan Overholt's MATRIX [1] and the "continuous keyboard" by Freed & Avizienis [2]. What they have in common is a large number of identical sensors that would otherwise require a large array of microcontrollers. But their technology is very specific to the instruments and not available as a separate interface. The latter, however, is related

1. The preferred method though is the use of a hardware description language like Verilog or VHDL

to CNMAT's Connectivity Processor [3], an interface that integrates multi-channel audio as well as GMICS[2] and a "gesture port".

The only other pure sensor interface with an FPGA is surprisingly one of the first at all - STEIM's SensorLab [4]. For its time it offered remarkable features such as its own programming language SPIDER that was particularly well suited to event handling. It also supported special sensors like e.g. ultrasound distance measurement and was able to drive character displays over a synchronous serial link. While the first two revisions implemented this additional functionality with discrete TTL logic, rev. C combined a 80C535 microcontroller with a Xilinx LCA[3] to accomplish this task and expand on it.

However, this rather complex design came at its price, so in the following years simpler MIDI interfaces were released that focused on analog inputs and simple digital I/O. Another trend was that MIDI was used less and less to control hardware synthesizers and samplers, but was fed directly into computers as those became powerful enough for realtime software synthesis. This also meant that the controller no longer had to perform event processing as this was a comparably easy task for the computer, where it could also be implemented in a much more flexible way. The downside of this approach is that sensor data has to be streamed continuously which can be quite taxing for a slow interface like MIDI. Consequently the move to USB, Ethernet [5], and even digital audio [6].

3. TOWARDS FPGAS

3.1 Concurrency and Precision

If one dispenses with event processing like outlined above this also means that there is actually little left to do for the microcontroller's CPU in terms of performing arithmetical and logical operations. Tasks like signal filtering, threshold detection, or even simple scaling can all be done on the host now, so the microcontroller effectively becomes a data pump where its main job is now to configure, control, and query the individual I/O subsystems. The most common technique here is the use of interrupts that signal e.g. the completion of an analog-to-digital conversion or the successful transmission of a data byte/packet.

This approach can be managed comparably easy as long as there are not too many subsystems involved and interrupt service routines can be kept short. However, in the scenario proposed in the introduction, where more and more I/O modules are added, the overall timing can become critical as different interrupts collide and one has to deal with priorities and more complex scheduling strategies.

The situation gets worse when the desired functionality has to be implemented under program control. E.g. if an array of PWM outputs is required but only a single counter module is left, rather than writing the pulse-widths into their separate registers (and then "forget" them) the main process constantly has to read this one counter and perform comparisons to the PWM values stored in data RAM (using the ALU[4]), before it can set or clear the corresponding output pins accordingly. This sequential scheme alone results in a reduced accuracy of the PWM signal; it gets far worse when it is interrupted by other activities..

Now, how does an FPGA, or more specifically the gluion, handle this challenge. To begin with, there simply is no program flow to be interrupted. While it would not be a problem to implement a CPU inside an FPGA (including program counter, program memory access, ALU, instruction decoder, execution unit, etc.), it simply is not necessary. Instead all modules operate concurrently and communicate thru registers or dual-ported RAM. In other words, there are no shared resources that have to be arbitrated. Every PWM of the above array can have its own counter and pulse-width register, which are compared continuously and not just when the CPU gets to the matching program location. As a result the signal is accurate down to a single cycle of the master clock, 50ns in the case of the gluion. This may seem like overkill, but it allows the implementation of high-frequency PWM while maintaining high resolution.

3.2 Fine-tuning and unique functionality

Not only can an FPGA host a large number of precise I/O modules, it also allows the user to fine-tune their parameters. While counters and registers inside microcontrollers are usually fixed at 8 or 16 bit resolution, with an FPGA it is just as easy to have a 3-bit counter to drive an 8-way multiplexer, or a 50-bit counter to cover days. This allows the designer to balance the resources of the specific chip being used.

Furthermore, originally simple modules can be enhanced with functionality not available in most microcontrollers. In the case of the PWM example introduced above, this could be dynamic control of the frequency or a 'burst' register to specify a defined number of pulses to be sent rather than the usual continuous stream. Effectively this becomes a basic stepper motor control. More examples are discussed in chapter 5.

Finally, the FPGA allows for unique modules not found in any microcontroller, enhanced or not.

3.3 Portability

When porting an existing design to a new microcontroller-based interface it is rather likely that it will have a different chip than the previous one. This will possibly be from the same chip family or at least the same vendor, although sometimes the latter has to be abandoned when more drastic changes are required. In any case, the assumption that C-code should be easily portable is not entirely applicable here. Considering the above scenario where the microcontroller is freed from event processing and is mainly busy managing I/O registers, the programmer cannot simply transfer the existing C-code but has to map the old I/O register set to the new one, with all its pecularities. Similarly, functionality that has been implemented under program control has to be thorougly reviewed to match new timing constraints.

In the FPGA-case, this is less of a problem as we have defined our registers ourselves. In fact, the main feature of new FPGA releases is their bigger size. Exceptions are e.g. RAM blocks or PLLs[5], but these are ususally generic enough, so the design can be easily adapted. Admittedly at the current state this is somewhat theoretical though as it is not backed by extensive experience.

Finally, it should be noted that by defining the very details of a

2. A bi-directional interface for audio and sensors proposed by Gibson
3. LCA = Logic Cell Array, conceptually similar to FPGAs
4. ALU = Arithmetic Logic Unit

5. PLL = Phase Locked Loop

chip's architecture it is possible to apply the concepts of open-source software to hardware designs, with all their socio-economic implications.

4. DESIGN OF THE gluion

In designing the gluion's hardware one of the goals was to use as few external components as possible. This was particularly true for the Ethernet-based host link. Following the above portability goal, no external Ethernet-PHY (physical layer) chip was used with all the lengthy and device-specific setup procedures. Instead this functionality was implemented entirely as a soft core, so for this part the only external components are the RJ-45 connector and the isolation transformer. On the downside, only 10base-T could be achieved, but this is usually enough for sensor data.

On top of this lie rudimentary data and transport layer modules. Rather than going for a complete TCP/IP stack only those protocols are implemented that are necessary for OSC, i.e. UDP, ARP, and IP. The choice for OSC [7] was for its high speed, powerful protocol, and driver/OS-independency.

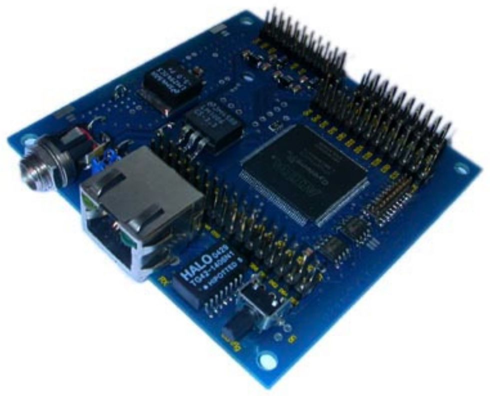

Figure 1: the gluion 'barefoot'

Apart from the I/O modules described above and in the following chapter, the gluion provides functionality to upload a new configuration via Ethernet. This means additional flexibility as the interface can be adapted when moving from project to project. Users can enter their requirements into a web interface which generates a script that drives the build process of the FPGA design tools. At the current state this is not yet fully automated though.

5. EXAMPLES FOR MODULES

5.1 Scanned Switch-Matrix

While connecting a switch to a sensor interface is a simple task with just about any sensor interface (usually a pull-up resistor and the switch/button towards ground), setting up a matrix of switches requires a multiplexed approach with both in- and output pins.

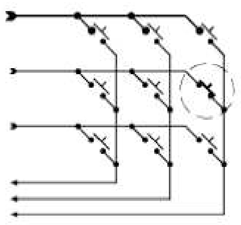

Figure 2: 6 I/O lines required for 9 buttons

The latter have to send synchronized pulses into the rows of the matrix, while the columns are connected to the inputs that listen for returned pulses which can then be matched to specific switches. The advantage of this concept is that less I/O pins are required compared to the pin-per-switch approach (approx. $2\sqrt{N}$ vs. N). As an example consider an interactive installation with a grid of 30x30 floor switches - the ordinary approach would ask for 900 pins, which is clearly beyond any available interface. In contrast, the gluion with its 68 digital pins can easily accommodate the 60 connections required for such a matrix.

5.2 Rotary Encoder

This component can be frequently found in commercial music gear as well as other industries where it is used to adjust values that are under software control. In contrast to a potentiometer it has no stops but can be turned "endlessly". In doing so it outputs a dual stream of pulses that can be counted while their phase relationship signals the direction of the rotation.

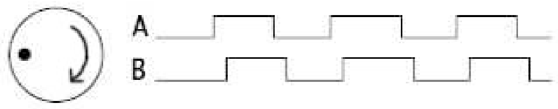

Figure 3: dual pulse trains from an encoder

A microcontroller could do this task under program control, but none of the reviewed sensor interfaces do this. It also becomes difficult to capture all pulses when using high-resolution encoders that create hundreds of increments per 360º turn. In an FPGA a simple state machine connected to a counter ensures that no pulse nor phase is lost.

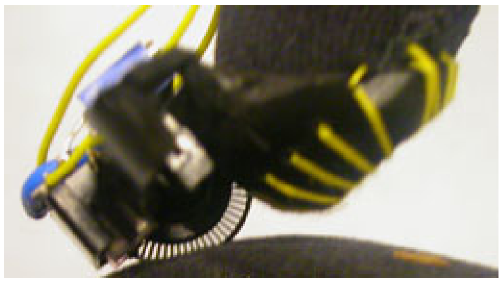

Figure 4: optical encoder from an old mechanical mouse mounted on the fingertip of a data glove

5.3 Ultrasound Distance Sensor

One way to measure distances is to send out an ultrasound pulse and wait how long it takes to reach the receiver. Three timers/ counters are required here: one to generate the 40kHz signal that common transducers operate at, one to count to ~10 for a small burst, and finally the actual time lag counter whose value is proportional to the distance covered.

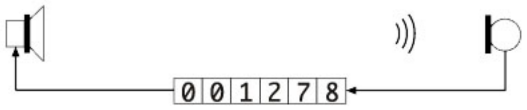

Figure 5: measuring distance by timing ultrasound lag

As the time lag counters are driven by the 20MHz master clock the internal resolution is 50ns equivalent to 0.017mm. The effective resolution, however, is limited by the thresholding circuit and is about a magnitude less. But compare this to the 1ms resolution achievable (at best) when doing the timing on the host machine, which translates into roughly 30cm.

Some ultrasound designs built for more common sensor interfaces actually have their own microcontroller to determine a distance value, which they then have to convert into a voltage to be measured by the interface's ADC. These additional steps obviously limit the achievable resolution and are hardly elegant.

5.4 Frequency Measurements

A frequency counter is ususally just a counter that is clocked from an external signal for a certain period called the gate time (which requires another counter). It can be used for sensors where the sensing element is part of an oscillator whose frequency changes depending on the physical parameter. Of course, frequencies can be converted to a voltage, however this adds to the complexity. Moreover, the frequency can be sent over long lines with less signal degradation than a voltage. In the example below a low-cost inductance sensor has been fitted on a tuba to measure its key positions.

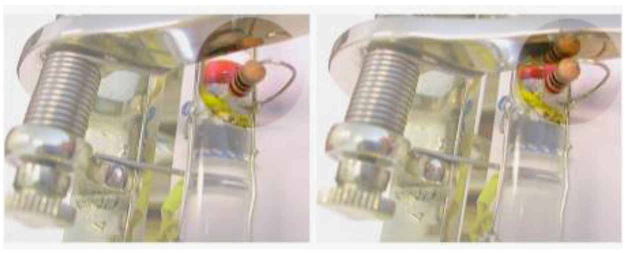

Figure 6: left: coil free-standing, right: coil covered by tuba key

In addition to the single-channel frequency module the gluion also has a multiplexed version that is e.g. used to measure the 61 key heights of the SKRUB keyboard [8].

5.5 Pulses

For signals with low frequencies it is often better to measure their period. The process is complementary to the above frequency measurement, as the external signals provides the gate for a counter driven by the internal clock.

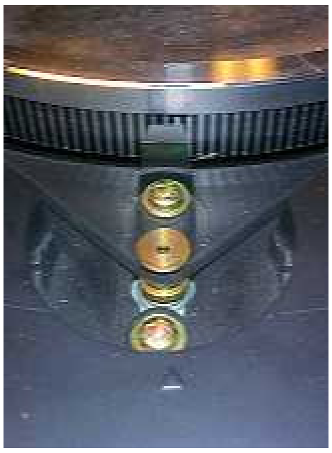

Figure 7: optical sensor and motor of a belt-driven record player

In the upper part of the above picture an optical sensor measures the duration of black bars passing by as the turntable rotates, effectively determining its speed. Pulses also flow in the other direction as a PWM signal controls the motor located in the lower part of the image.

5.6 Serial Data

While the above modules allow the direct control of many different sensors and actuators there are often existing devices that need to be tied into the overall system. Wether it is a Polhemus 6DOF tracker, a character display, or a DMX lighting setup, they usually communicate thru a serial interface. For this it is possible to configure modules for various serial protocols. Currently available are RS232, MIDI, DMX, and SPI. I2C and PS/2 are planned.

6. FUTURE PLANS

Currently wireless transmission of sensor data is possible through a simple WiFi bridge. However, a more integrated solution is being considered to keep the overall design compact.

Several enhancements to the existing I/O modules are underway as are entirely new modules like a touchscreen controller.

Other plans call for more hardware modules, like ultrasound amplification or drivers for motor control.

Finally a move to a bigger FPGA might be necessary to host more complex designs.

7. ACKNOWLEDGMENTS

I would like to thank Podewil (Elke Moltrecht) as well as its successor Tesla (Carsten Seyfarth) for their continued support in hosting the glui lab. Also thanks to Nic Collins and Reinhold Friedl.

8. REFERENCES

[1] D. Overholt, The MATRIX: A Novel Controller for Musical Expression, In *Proceedings, CHI '01 Workshop on New Interfaces for Musical Expression (NIME'01), Seattle, WA,* 2001.

[2] A. Freed and R. Avizienis: A New Music Keyboard featuring Continuous Key-position Sensing and High-speed Communication Options, In *Proceedings, International Computer Music Conference, Berlin, Germany,* 2000.

[3] A. Freed, R. Avizienis, T. Suzuki, and D. Wessel: Scalable Connectivity Processor for Computer Music Performance Systems, In *Proceedings, International Computer Music Conference, Berlin, Germany,* 2000.

[4] http://www.steim.org/steim/sensor.html

[5] T. Coduys, C. Henry, and A. Cont: TOASTER and KROONDE: High-Resolution and High-Speed Real-time Sensor Interfaces, In *Proceedings, New Interfaces for Musical Expression (NIME-04) Hamamatsu*, 2004.

[6] J. Allison and T. Place: Teabox: A Sensor Data Interface System, In *Proceedings, New Interfaces for Musical Expression (NIME-05) Vancouver*, 2005.

[7] M. Wright and A. Freed: OpenSound Control: A New Protocol for Communicating with Sound Synthesizers, In *Proceedings, International Computer Music Conference, Thessaloniki, Hellas,* 1997.

[8] http://glui.de/prod/gluiph/SKRUB.html