MANAGING MULTIPLE FUNCTION GENERATORS FOR FAIR

S. Rauch, M. Thieme, R.C. Bär, GSI, Darmstadt, Germany

Abstract

In the FAIR control system, equipment which needs to be controlled with ramped nominal values (e.g. power converters) is controlled by a standard front-end controller called scalable control unit (SCU). An SCU combines a ComExpressBoard with Intel CPU and an FPGA baseboard and acts as bus-master on the SCU host-bus. Up to 12 function generators can be implemented in slave-board FPGAs and can be controlled from one SCU.

The real-time data supply for the generators demands a special software/hardware approach. Direct control of the generators with a FESA (front-end control software architecture) class, running on an Intel Atom CPU with Linux, does not meet the timing requirements. So an extra layer with an LM32 soft-core CPU is added to the FPGA. Communication between Linux and the LM32 is done via shared memory and a circular buffer data structure. The LM32 supplies the function generators with new parameter sets when it is triggered by interrupts. This two-step approach decouples the Linux CPU from the hard real-time requirements. For synchronous start and coherent clocking of all function generators, special pins on the SCU backplane are being used to avoid bus latencies.

DESCRIPTION OF SCU AND FG

The quadratic function generator (FG) which is described in this paper, is a VDHL macro that runs in SCU bus slave cards. At the moment, there are three slave cards with this feature: DIOB (1 FG), ADDAC1 (2 FGs) and ADDAC2 (2 FGs). The DIOB card has an digital output with 32 Bit for the FG output value. The two ADDAC cards offer an analog output with 16 Bit resolution for the FGs. The slave cards are controlled via the SCU bus from the Scalable Control Unit (SCU). The SCU is a FPGA based controller equipped with a ComExpress Board which runs linux. The communication between FPGA and ComExpress Board is done via PCIe. Inside the FPGA is a System-on-Chip (SoC) on basis of a wishbone [1] crossbar with a PCIe-to-wishbone bridge (wishbone master). The SCU bus is connected with a bridge too, that acts as a wishbone slave. Part of the SoC is LM32 cluster with a configurable number of softcore processors and shared memory (see Fig. 1). A separate crossbar is used for message signaled interrupts (MSI). A interrupt master, e.g. SCU bus bridge, sends MSIs to an interrupt slave. That can be a interrupt queue of an LM32 or the PCIe bridge. With the use of MSIs, the interrupt system is quiet flexible, because every slave can address every interrupt target.

The SCU bus is a parallel bus with 12 slave slots. The data and address lines are each 16 Bit wide. Each slave has a separate IRQ line. Inside the SCU bus bridge the IRQs are translated into MSIs. The system should be used as an arbitrary function generator with 12 independent channels. Each channel will control equipment that needs ramped nominal values. That means for example power converters and Direct Digital Synthesis (DDS) systems. The FG is configured with a set of data and interpolates then a predefined number of output values. After the interpolation is started, the FG waits for the next set of data that is provided by the linux FESA class. A brief hardware description of the FG can be found here [2]. In contrast to the older paper, a few things had to be changed for the implementation. The data path is now 64 Bit wide and both parameters, the linear and the quadratic one, can now be shifted in a 64 Bit range.

FG Inside SCU

Other then for ramped power converters, the SCU will be used to control DDS systems. This will be done with FIB cards, which are supplied from the radio frequency group. These FIB [3] cards are used as SCU slaves. But in there current revision the are not able to run a FG macro in the slaves. So a different solution had to be found. The same FG macro as used in the slaves is put behind a wishbone interface and connected to the crossbar of the SCU. The output of that FG is connected to a special wishbone master, that splits the 32 Bit output value from the FG into two 16 Bit accesses to the SCU bus. So the interpolation of the FG is done inside the SCU, instead of inside the SCU slaves. Because this modus uses a lot more bandwidth than the slave approach, the SCU bus should only be used for sending FG values. The FG macro with wishbone interface acts exactly like the FG in the slaves, only the interface is different. For the software layer they look identical.

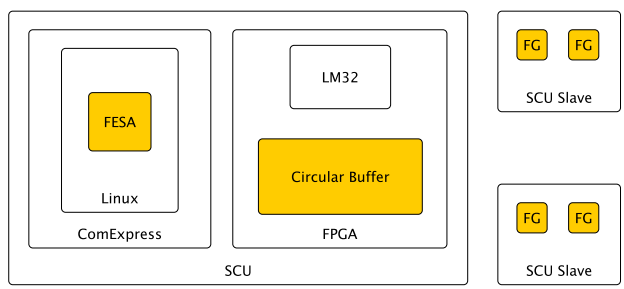

Figure 1: SCU with two slaves.

DATA SUPPLY WITH REAL TIME BOUNDARIES

The FG can be configured to interpolate in steps from 250 up to 32000. The sample frequency is configurable from 16 kHz up to 1 MHz. If the FG should now sample with 1 MHz for 250 steps that means the linux program has to provide a new data set every 250 µs. This data rate is to high for linux to be serviced reliably for 12 channels.

ISBN 978-3-95450-146-5

IMPLEMENTATION WITH LM32 AND MSI

As direct control of the FG macro from linux is not possible, another real time layer is inserted. One of these LM32s is now used for configuring the FGs and supplying new data when triggered by the FG. Because each of the LM32 softcores has a dual port memory, that is accessible for the other LM32s and the linux system via the PCIe bridge, communication is done via circular buffers in shared memory. The software running on the LM32 is written in C and needs no operating system. So the real time behavior is easy to predict. The real time processing is done in a IRQ handler, that supplies the FGs with data. The round trip time for each interrupt from the source (FG macro) to the IRQ handler and then back again is around 5µs. One SCU bus access is 300ns long. With six SCU bus accesses needed for every parameter set, that makes 18µs for the handling of each FG. With a step width of 250 theoretically 13 FGs could be serviced. In reality the smallest step width used, will be 1000. That gives enough time for handling the 12 FGs needed.

For controlling the LM32 software, another IRQ handler is used. Here the linux software generates MSIs by writing to LM32 interrupt queues. This works as a software interrupt scheme.

FG OPERATION

After reset the LM32 software scans the SCU bus and enumerates the FG macros found in the slave cards. In addition Up to 12 FGs are supported. These virtual FG devices are then presented to the FESA class. A virtual device number (0-11) is stored in each FG to make it easer to address the right FG, when an interrupt occurs. Each virtual device has its own circular buffer, that is filled by the FESA class and emptied by the IRQ handler. When the buffer is empty, the sampling of the FG halts.

The software sends the first parameter set with the start value to each FG. After that, the FGs can be started with a broadcast write to the SCU bus or with a signal from a timing event. Directly after the start, the FG send a data request for the next parameter set. The signaling of the data request is done with the IRQ feature of the SCU bus and the MSI system of the SCU. Encoded in the MSI message is the number of the slave card, that has triggered a data request IRQ. Because there can be two FG macros in one slave card and the slaves use a shared IRQ scheme, the handler has to ask the slave, which macro has triggered the data request. It then reads the virtual FG number, to select the right buffer for the FG. Then the handler sends the next parameter set from the buffer and acknowledges the IRQ.

PROJECT STATUS AND FUTURE WORK

The hardware implementation of the FG macro in both modes (SCU slave/Wishbone slave) is done. A test program exist under linux, that is able to start the FGs with arbitrary ramp data supplied by a text file. The integration into FESA is in progress.

A Data Aquisition (DAQ) system for the slaves is planned, that can make use of the same infrastructure as the FG. Only the direction of the data flow would be the other way around.

At the moment the data transfer between linux and the LM32 runs in polling mode. The linux program constantly tries to fill the circular buffer. This wastes lots of bandwidth. It would be more effective, if the LM32 signals to the linux program, that the buffer needs to be refilled. That can be done with an interrupt to the PCIe bridge.

REFERENCES

[1] Wishbone B4: http://cdn.opencores.org/downloads/wbspec_b4.pdf

[2] S. Rauch et al., "Improved Function Generator for Device Control for the GSI Control System", TUP006, http://jacow.org/pc08/papers/tup006.pdf, PCaPAC08, Ljubljana, Slovenia (2008).

[3] M. Kumm et al., "Realtime Communication Based on Optical Fibers for the Control of Digital RF Components", GSI Scientific Report (2007).

ISBN 978-3-95450-146-5