Embedded controller for GEM detector readout system

Wojciech M. Zabołotny[a] and Adrian Byszuk[a] and Maryna Chernyshova[b] Radosław Cieszewski[a] and Tomasz Czarski[b] and Wojciech Dominik[c] and Katarzyna L. Jakubowska[b] and Grzegorz Kasprowicz[a] and Krzysztof Poźniak[a] and Jacek Rzadkiewicz[b,d] and Marek Scholz[e]

[a]Institute of Electronic Systems, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warszawa, Poland;
[b]Institute of Plasma Physics and Laser Microfusion, ul. Hery 23, 01-497 Warszawa, Poland;
[c]Institute of Experimental Physics, Faculty of Physics, Warsaw University, ul. Hoża 69, 00-681 Warszawa, Poland;
[d] National Centre for Nuclear Research, ul. Andrzeja Sołtana 7, 05-400 Otwock, Świerk, Poland;
[e] Institute of Nuclear Physics, Polish Academy of Sciences, ul. Radzikowskiego 152, 31-342 Kraków, Poland;

ABSTRACT

This paper describes the embedded controller used for the multichannel readout system for the GEM detector. The controller is based on the embedded Mini ITX mainboard, running the GNU/Linux operating system. The controller offers two interfaces to communicate with the FPGA based readout system. FPGA configuration and diagnostics is controlled via low speed USB based interface, while high-speed setup of the readout parameters and reception of the measured data is handled by the PCI Express (PCIe) interface. Hardware access is synchronized by the dedicated server written in C. Multiple clients may connect to this server via TCP/IP network, and different priority is assigned to individual clients. Specialized protocols have been implemented both for low level access on register level and for high level access with transfer of structured data with "msgpack" protocol. High level functionalities have been split between multiple TCP/IP servers for parallel operation. Status of the system may be checked, and basic maintenance may be performed via web interface, while the expert access is possible via SSH server. System was designed with reliability and flexibility in mind.

Keywords: Data acquisition systems, GEM detectors, embedded systems, data processing, system control

1. INTRODUCTION

The block diagram of the GEM[1] detector readout system[2,3] is shown in the Figure 1. The main part of the readout system consists of the FPGA chips, which control the front end boards (FEB) and analog-to-digital converters (ADC), receive the data, preprocess them, detecting hits and estimating their energy and position, time-stamp the data using the external synchronization signal and store results in the DDR memory.[4,5]

This FPGA based system requires sophisticated configuration after power-up, and control when performing measurements. Finally the acquired data, after some processing, must be transferred to the data acquisition system via the computer network. To perform those functions the readout system is equipped with the dedicated controller based on embedded computer system.

2. REQUIREMENTS FOR THE SYSTEM CONTROLLER

The proposed system controller must meet wide range of requirements related to its desired functionality and reliability.

- Initialization of the system
  When the readout system is switched on, it is necessary to switch on power supply of different subsystems, checking their correct operation. Additionally it is necessary to configure the FPGA chips, using the configuration bitstreams.

Further author information: (Send correspondence to W.M.Z.)
W.M.Z.: E-mail: wzab@ise.pw.edu.pl, Telephone: +48 22 234 7717

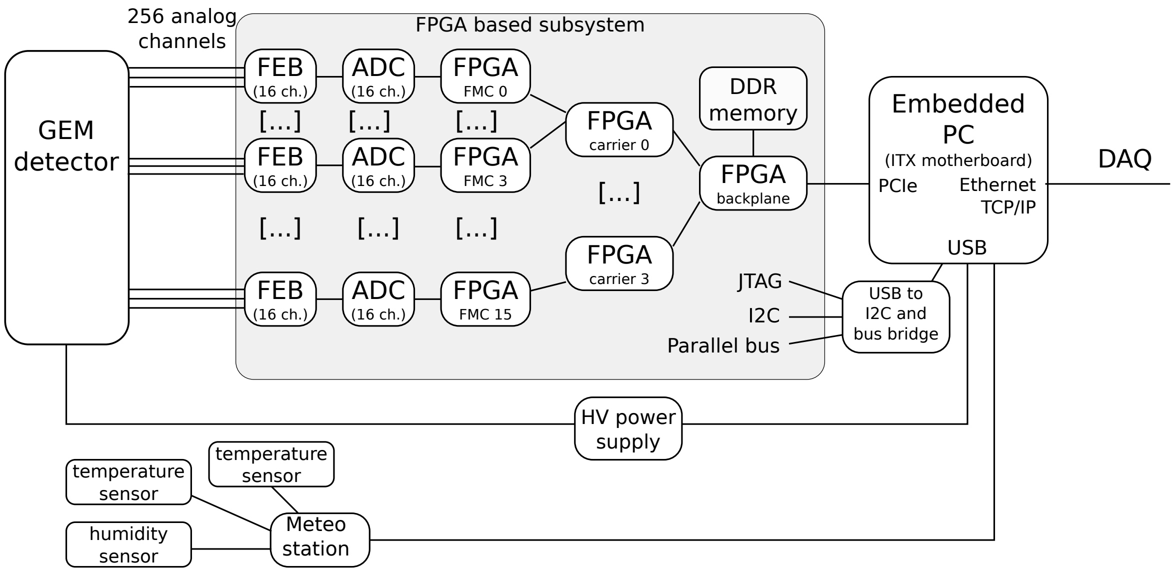


Figure 1. Block diagram of the GEM detector with readout system and controller.

- Monitoring of the system
  When system is properly initialized, it is necessary to continuously monitor its state, providing notifications about possible problems, and undertaking necessary actions to protect the system.

- Control of the system
  The GEM detector may be used in different modes. Usually it measures the radiation produced by the investigated object under control of the data acquisition system, but it must be also periodically calibrated with the known radiation provided by the calibration radiation source. Additionally sometimes it may be used in a special “development mode”, when the operator works with the detector interactively, or executing short scripts.

  This variety of different operating modes requires, that the system controller must be able to accept requests from different clients, assuring appropriate arbitration and synchronization of requests.

- Transmission of the measured data
  The data acquired by the FPGA based part of the system are stored in the DDR memory, connected to the FPGA chip. These data must be efficiently transferred to the system controller, submitted to further processing and finally transferred to the data acquisition system via computer network.

  Performing of these tasks requires availability of the high speed local link to quickly transmit huge amount of data from the FPGA connected DDR memory to the computer system, and possibility to work with the TCP/IP network.

Except of the above requirements related to the system’s functionality, there is yet another set of requirements related to the reliability of the system.

- Autonomous unsupervised operation
  The readout system is located near to the GEM detector, and may be difficult to access. Therefore it is important, that the system is able to work for a long time without supervision.

- Possibility of remote management
  In case if the system is hardly accessible, it should be possible to control most of its functions from remote location via the TCP/IP network.

- Robustness against random power failures
  It is possible that the system may experience random power cut, due to the power supply system malfunction or due to erroneous operation. It is necessary that such incident does not damage the system.

3. HARDWARE PLATFORM AND OPERATING SYSTEM

As the hardware platform fulfilling the requirements listed in the previous section, we have used the embedded PC, based on the ITX motherboard. Use of the standard hardware platform allowed us to choose the operating system from the broad range of operating systems available for PC-compatible systems.

To have full control over the operating system and drivers, we have decided for an Open Source solution and used the dedicated Linux system, built with the Buildroot[6] environment. The Buildroot environment allowed us to build the small Linux system with precisely selected set of programs and drivers needed to implement our controller. Additionally it allowed us to easily add additional software not included in the standard Buildroot environment.[7]

4. SOFTWARE ORGANIZATION

Basing on selected hardware platform and operating system, we could design the software architecture of our controller. According to the requirements described in section 2, our controller is supposed to work with multiple clients trying to perform certain actions with the hardware components of the readout system. To assure proper synchronization between operations performed by different clients, the direct hardware access is controlled by the dedicated "Main Hardware Server", responsible for proper serialization of requests.

4.1 Main Hardware Server

The Main Hardware Server is written in C, to assure precise control of resources utilization and to provide highly optimized implementation of hardware handling routines.

The Main Hardware Server controls the USB bridge,[8] providing access to different low-speed interfaces in the system, such as the I^2C bus, the JTAG interface used to configure FPGA chips, and the parallel local bus used to access the diagnostic registers.

The Main Hardware Server ensures also access to the PCI Express (PCIe) interface,[9] used to provide high-speed access to configuration registers in the FPGA based subsystem, and to transfer results of measurements from the DDR memory.

From the client's point of view, the Main Hardware Server implements a multithreaded TCP server, listening on the 54321 port. Each client is serviced in a separate thread, and a set of semaphores is used to avoid collisions when accessing different hardware components. Special care is taken to avoid memory leaks in a situation, where due to access errors one of client threads is terminated, e.g. due to erroneous attempt to access the hardware. As a special precaution, wherever possible, all data structures needed to execute particular request are allocated on stack, so they are freed automatically, when handling of the request is completed or terminated.

4.2 Implementation of higher level procedures

The Main Hardware Server provides access to the hardware components both for external clients, and for other, internal servers, responsible for performing of more complex operations. Those servers are needed to allow autonomous operation of the system.

To allow easy and fast development and debugging of high level procedures, they have been implemented in Python. Even though programs written in Python are executed significantly slower than equivalent programs written in C, Python offers important advantages:

- Automatic memory management. Risk of memory leaks due to errors in memory management is significantly reduced for Python programs, comparing to programs written in C or C++.
- Python is an object oriented programming language. The programmer may easily implement classes describing the hardware components, and create methods corresponding to typical operations performed with those hardware components.
- Python offers powerful and flexible exceptions mechanism allowing to precisely handle different errors, occurring during the execution of the program, e.g. due to errors in communication with hardware or communication via network. This functionality is very significant, when implementing hardware oriented systems, like this controller.

- Python offers a broad range of libraries solving typical programming problems. E.g. it is possible to implement a TCP server in just a few lines of code.

The higher level procedures have been implemented as a set of servers performing separate control or diagnostic tasks.

4.3 Monitoring and Measurement Server

The main server implementing the high level procedures is the Monitoring and Measurement Server. Its responsibility is to periodically check the state of the detector and of the readout system. Additionally it performs the calibration measurements of the radiation produced by the calibration ^{55}Fe iron source, to check current gain of the detector. On request from the central data acquisition system, this server also performs the measurement of the radiation produced by the investigated object - which is the most important task of the whole detector system.

Main problem in this server was to assure correct mode switching between the monitoring and calibration measurements and the measurements of the object. As the configuration of the hardware for those two modes may be different, it is necessary to perform reconfiguration of the hardware before the measurement. To allow that, the central acquisition system must provide the early notification, that the measurement of the object is planned in the nearest future. This notification interrupts the monitoring or calibration procedure, starts configuration of the system for object measurement, and forces the system to wait for the hardware trigger system, starting the measurement.

To minimize the required time between the early notification and the measurement, it is desirable, that the interruption of the monitoring or calibration procedure should be as fast as possible, however this is not easy to achieve. Not all control procedures may be interrupted on any time, and some of them, when not completed properly may leave the system in an unstable state. Therefore it was necessary to implement a sort of cooperative preemptivity. The monitoring ans calibration procedures actively check in appropriate moments, if there is a request to switch into the measurement mode.

The Monitoring and Measurement Server is accessible via TCP/IP network at two ports. The port 54325 is available for requests related to performing calibration and measurements, while port 54324 is available for sending of new configuration settings for the detector and readout system. Such separation of ports for different services allowed to simplify the protocol and the server.

Two other servers are managing the external hardware connected to the readout system via USB interface.

4.4 Meteo station server

As the parameters of the GEM detector depend on such parameters as temperature and air pressure,[10] a USB connected meteo station is provided to continuously monitor those parameters and additionally humidity. As the meteo station sends measured parameters to a single receiver, a dedicated server is implemented in Python, which receives the results from the meteo station, stores them, and on request forwards the last result to the client connected to the 54323 TCP/IP port. Multiple clients may be served in parallel.

4.5 High voltage server

For correct operation, the GEM detector requires the High Voltage (HV) power supply. This voltage must be precisely controlled, and erroneous setting of too high voltage may result in a permanent damage of the detector. The HV power supply used in our system is controlled via the USB interface. In case if multiple clients are trying to contact the HV power supply (e.g. the first one trying to set new voltage settings, and the second one requesting results of measurement of current output voltages), possible mixing of their request could have disastrous consequences. The HV server receives requests from multiple clients connected to the 54322 port, and handles them consecutively. The HV server is also implemented in Python.

5. NETWORK PROTOCOLS

The readout controller uses the TCP/IP network to communicate with clients. The important assumption, we have made, is that the system is connected to the physically protected private network, which is usually the case in professional data acquisition systems. This assumption allowed us to implement very simple network protocols without the overhead related to security issues associated with usage of publicly available networks.

Anyway due to the use of Linux OS, if it is necessary, it is trivial to add an additional layer in the network protocols (e.g. SSL/TLS, or VPN), which would cryptographically protect the controller, providing authentication of clients and control of integrity of transmitted commands and responses.

5.1 Protocols used by the Main Hardware Server

The Main Hardware Server is listening at 54321 TCP/IP port and uses two protocols. The first one, the “basic protocol” implements very simple operations, like sequence of read or write accesses via the local bus, via the I^2C bus, or via PCIe bus. Additionally the basic protocol implements requests to configure the particular FPGA chip with the SVF file transmitted to the server. The syntax of requests in the basic protocol is very simple. Each request contains the type of request, the length of the request and associated data, and the data. Execution of each request is confirmed by the server. In case of reception of erroneous or corrupted request, the server notifies the client with an error message, and additionally closes the connection, to prevent further interpretation of corrupted input data, which could result in execution of random requests. The client in this case should handle the error and reconnect to the server.

The basic protocol is used by other servers, and also by external Matlab based clients, which are used for development of algorithms for control of the detector, analysis of data, and calibration of the detector[11, 12]

The basic protocol implements also a special command allowing the client to switch the server to the second one – “extended protocol”.

The extended protocol is used for more complex commands and procedures, which require sending of more complex, structured request arguments to the server, and receiving complex, structured results. When implementing the extended protocol, many solutions like standard RPC, or JSON were considered, but finally we have decided use the msgpack[13] library. The msgpack library allows communication between clients and servers written in different languages, e.g. Java, C, C++, Python and many others and offers efficient packing of different data formats.

5.2 Protocols used by the Measurement and Monitoring Server, HV Server and Meteo Server

The Measurement and Monitoring server uses the msgpack based protocol for requests related to calibration and measurements.

However, as the configuration settings are prepared, and sent by the Matlab based system, and there is no reasonable msgpack implementation for Matlab yet, the requests related to configuration of the readout system use another protocol, based on sending of text messages in format compatible with the ConfigParser[14] Python package.

Similarly the protocols used by the Meteo Server and HV Server are based on simple text messages to allow easy communication with Matlab based clients. In case of the HV Server, the transmitted messages are additionally protected with the checksum, to limit risk of inadvertently setting of incorrect high voltage value due to corrupted message.

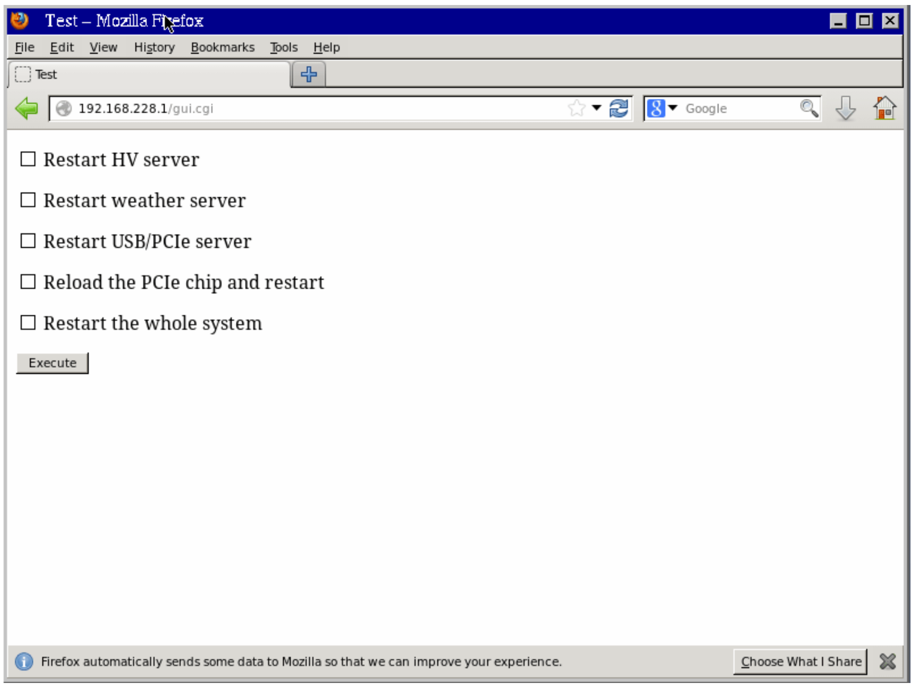
6. DIAGNOSTIC FUNCTIONS

The important part of activity of the Measurement and Monitoring Server is the continuous monitoring of the state of the readout system. It is possible due to availability of multiple sensors, measuring the voltages, currents and temperatures in different parts of the system, connected to the I^2C bus.

To allow easy acquisition and handling of these diagnostic information, a dedicated, object oriented I^2C framework has been implemented in Python. This framework allows to create and maintain two independent hierarchies of the I^2C sensors. The first one - related to the position of the sensor in the I^2C bus. It automatically sends correct values to the I^2C switches in case of the multi-branch I^2C bus, as needed to access the particular sensor. The second one - reflects the logical organization of the sensors. The framework automatically detects damaged or inaccessible sensors. When a report about the state of the system is requested, the first hierarchy is scanned to access all sensors, and then the second one is used to organize the acquired information. The final report is generated in the XML format, which is both - human readable and machine readable, and may be both - included in the results of measurement, and written to the logs of the system, as described in section 9.

7. REMOTE HANDLING

The controller mainly works communicating with remote clients via TCP/IP network. However sometimes it is necessary to remotely perform some maintenance operations. In this case the operator may use the HTTP server, which is implemented in Python, and offers graphical user interface, allowing to perform simple maintenance (see Figure 2).



☐ Restart HV server

☐ Restart weather server

☐ Restart USB/PCIe server

☐ Reload the PCIe chip and restart

☐ Restart the whole system

Execute

Firefox automatically sends some data to Mozilla so that we can improve your experience. Choose What I Share

Figure 2. Access to the basic maintenance procedures via WWW browser and embedded HTTP server.

The HTTP server is also used by remote clients to obtain information, that the system has completed initialization after power-up and is ready to work (see Figure 3).

If more advanced intervention in the system is needed, the expert may log in via the SSH server, and perform necessary actions from the system shell command line. Additional measures, described in section 8, are provided to allow testing of the modified version of system software, with possibility to quickly restore the standard version.

In the worst case of system instability, if even the network connection and/or SSH server do not work, it is possible to connect the serial console (which also allows remote operation, if connected to the serial port of another network connected system), or standard keyboard and LCD display, to diagnose the system locally.

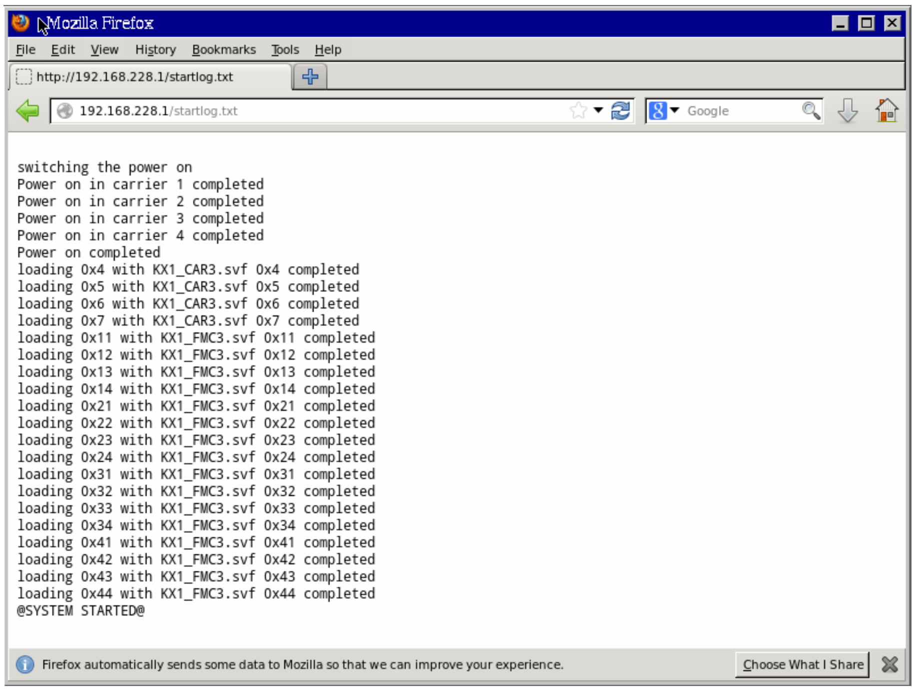
8. RELIABILITY AND FLEXIBILITY OF THE DESIGN

In current embedded systems, the filesystem layer intensively uses data buffering to improve performance, resulting in a risk, that inappropriate shutting down of the system e.g. due to power failure or incorrect operation, may lead to filesystem corruption and may render the system unusable.

This risk may be significantly reduced by use of journaling filesystems, however in our system we have decided to use yet another solution. Due to low price of RAM memory, it was possible to equip the hardware platform with memory of capacity sufficient, to load the whole operating system image into the ramdisk. Namely we used the ITX board equipped with 4GB of DDR3 RAM.

The controller system is booted from the external USB flash disk. During the start up of the system, the Linux kernel together with the initramfs root file system is loaded from that disk into the memory. After the system is booted, all programs are available in the ramdisk. Such solution speeds up operation of the system, as there is no need to load programs from relatively slow mass storage during the operation. Of course this approach is suitable mainly for embedded systems with limited and carefully selected set of programs - just like our system created with the Buildroot environment.

With this solution the external USB flash disk could be mounted read-only, preventing any possibility of corruption due to random power failure. However sometimes it is necessary to modify the information stored in the flashdisk - therefore it is mounted in read-write mode, but all modifications are performed in such a way, that the new file to be written into the


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.228.1/startlog.txt
192.168.228.1/startlog.txt
Google

```
switching the power on
Power on in carrier 1 completed
Power on in carrier 2 completed
Power on in carrier 3 completed
Power on in carrier 4 completed
Power on completed
loading 0x4 with KX1_CAR3.svf 0x4 completed
loading 0x5 with KX1_CAR3.svf 0x5 completed
loading 0x6 with KX1_CAR3.svf 0x6 completed
loading 0x7 with KX1_CAR3.svf 0x7 completed
loading 0x11 with KX1_FMC3.svf 0x11 completed
loading 0x12 with KX1_FMC3.svf 0x12 completed
loading 0x13 with KX1_FMC3.svf 0x13 completed
loading 0x14 with KX1_FMC3.svf 0x14 completed
loading 0x21 with KX1_FMC3.svf 0x21 completed
loading 0x22 with KX1_FMC3.svf 0x22 completed
loading 0x23 with KX1_FMC3.svf 0x23 completed
loading 0x24 with KX1_FMC3.svf 0x24 completed
loading 0x31 with KX1_FMC3.svf 0x31 completed
loading 0x32 with KX1_FMC3.svf 0x32 completed
loading 0x33 with KX1_FMC3.svf 0x33 completed
loading 0x34 with KX1_FMC3.svf 0x34 completed
loading 0x41 with KX1_FMC3.svf 0x41 completed
loading 0x42 with KX1_FMC3.svf 0x42 completed
loading 0x43 with KX1_FMC3.svf 0x43 completed
loading 0x44 with KX1_FMC3.svf 0x44 completed
@SYSTEM STARTED@
```

Firefox automatically sends some data to Mozilla so that we can improve your experience.

Figure 3. Logs from the start of the system, available via the HTTP server,

```
#!/bin/sh
#/etc/init.d/S41overlay file
modprobe vfat
mkdir /tmp/d
#Mount the boot USB flashdisk in /tmp/d
mount /dev/sda1 /tmp/d
#If the overlay script is present,
#execute it
if [ -e /tmp/d/overlay.sh ] ; then
  /tmp/d/overlay.sh
fi
```

```
#!/bin/sh
#Sample overlay.sh file to be placed
#in the main directory in USB flashdisk
if [ -e /tmp/d/overlay.tbz2 ] ; then
(
  cd /
  cat /tmp/d/overlay.tbz2 | bzip2 -cd | tar -xf -
)
fi
```

Figure 4. Sample shell scripts involved in process of modification of standard operating system image during booting of the system.

flashdisk is prepared in advance, and then quickly copied to the flashdisk, with the “sync” command executed immediately after the copying is finished. That way the time window, in which random power failure may lead to filesystem corruption is reduced to minimum. Of course it is possible to further reduce this danger by use of journaling file system on the flashdisk, but to maintain compatibility with Windows machines, we have decided to use flashdisk formatted with the standard VFAT filesystem.

The described approach facilitates also testing and debugging of the controller itself, as it is very easy to modify some user programs in the ramdisk. In case if such modification leads to system instability, it is sufficient to reboot to recover standard software and correct operation.

8.1 Flexibility of the design

The operating system image (kernel + initramfs data) produced by the Buildroot environment may be quite big file (it is a 40MB file in current version of our controller), and its recompilation may also consume significant amount of time. Therefore, to make small modifications easier, we have added possibility to add a special shell script in the main directory of the boot USB flashdisk. During booting of the system existence of this script is checked, and if it is present, it is executed on very early stage of system initialization. Typically, this script is used to unpack the tar archive containing modified versions of programs and scripts. Sample content of scripts involved in that process is shown in the Figure4

To make sure, that essential services are running, whenever the server is started, the servers are started by the *init* process, according to the definitions in the */etc/inittab* file. Such solution however makes difficult to test modified version

```
# Part of the /etc/inittab file
# responsible for starting the
# monitoring server
# now run any rc scripts
::sysinit:/etc/init.d/rcS
# [...]
tty5::respawn:/opt/srv_monitor.sh
```

```
#!/bin/sh
# /opt/srv_monitor.sh wrapper script,
# responsible for running
# the measurement and monitoring server
while true; do
  if [ -e /var/run_monitor ]; then
    /opt/srv_monitor.py
    # if server crashes, try to restart it
    # after 5 seconds
    sleep 5
  else
    # If start of the server was forbidden,
    # check again after 10 seconds
    sleep 10
  fi
done
```

Figure 5. The inittab configuration and shell script involved in control of running of monitoring server. Similar solution is used do control starting of other servers.

of the particular server . Therefore starting of each server is enabled only if a special file is present in the */var* directory. Details of the implementation are shown in the Figure 5.

With such implementation the operator may delete the file */var/run_monitor*, kill the running server, and start the modified version or run other program controlling the related part of the hardware. Again, simple rebooting of the system recovers standard operation.

8.2 Flexible configuration of the FPGA chips

The controller provides also means to easily modify configuration of the FPGA chips. These chips are configured during the start-up of the system with the SVF files, stored in the USB flash disk. Except of the SVF file themselves, there is also stored information about their internal registers structure, according to the Internal Interface[15, 16] specification. All those files may be easily replaced with the new or experimental versions of the FPGA firmware.

It is also possible to keep the standard version of those files in a backup directory on the same disk, making restoration of the standard configuration very easy.

9. LOGGING OF POSSIBLE ERRORS AND EXCEPTIONS

The readout system requires continuous monitoring of the state of detector and the system itself. Any abnormalities or execution errors should be logged for further analysis. The decision to build a system controller not equipped with a hard disk, disallows us to use the standard approach with logs written to the dedicated directory in a filesystem on harddisk.

Available options were as follows:

- Writing logs to the directory in the ramdisk. Of course in this approach we must limit the size of the logfiles, as filling the ramdisk with the logfiles may induce system instability. Fortunately, the Python package *logging*[17] offers functionality to rotate the logfiles when they reach predefined size. The big disadvantage of such approach, that in case of failure causing system crash, it is impossible to read and analyze the logfiles.
- Writing logs to the logfiles located in the FLASH disk. In this approach we also must to limit the size of the logfiles due to limited capacity of the flashdisk. Because the main filesystem in the flashdisk should be mounted read-only due to reliability reasons, as described in section 8, we need to provide another partition on the FLASH disk for logfiles. Unfortunately such multi-partition USB flash disks are not compatible with the Windows operating system, which recognizes only the first partition. Therefore analysis of logs after hypothetical system crash would not be possible on the Windows machine. Another variant of this approach could be to provide one USB flash disk for system initialization and configuration data, and the another one for system logs. Additionally the “log disks” could be easily replaced with the empty ones when they get filled with the log data, and archived.

  There is yet another disadvantage of this approach however. To limit the wear of the FLASH disk, it is desirable to buffer the log information in the memory and write it to the disk only when a complete buffer is filled. Unfortunately this data buffering results in loss of the last log data in case of serious stability problems resulting in kernel panic.

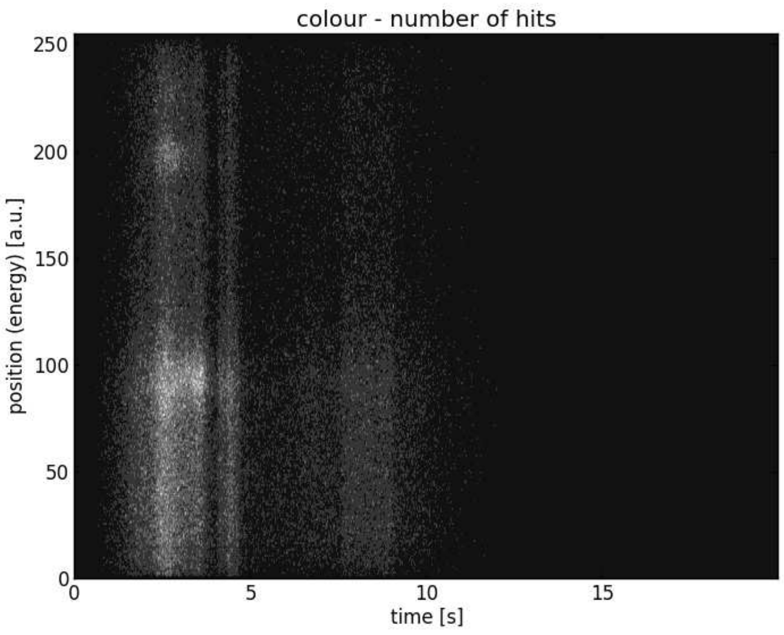


Figure 6. Example of the X-ray spectrogram of plasma, measured during the tests in the JET facility. The X-axis is the time in seconds. The Y-axis is the position of the hit in the 1-D GEM detector, which corresponds to the photon energy (in arbitrary units). The colour of the pixel shows the number of hits.

So the most interesting data, which may be extremely useful when investigating causes of the experienced problems are likely to be lost.

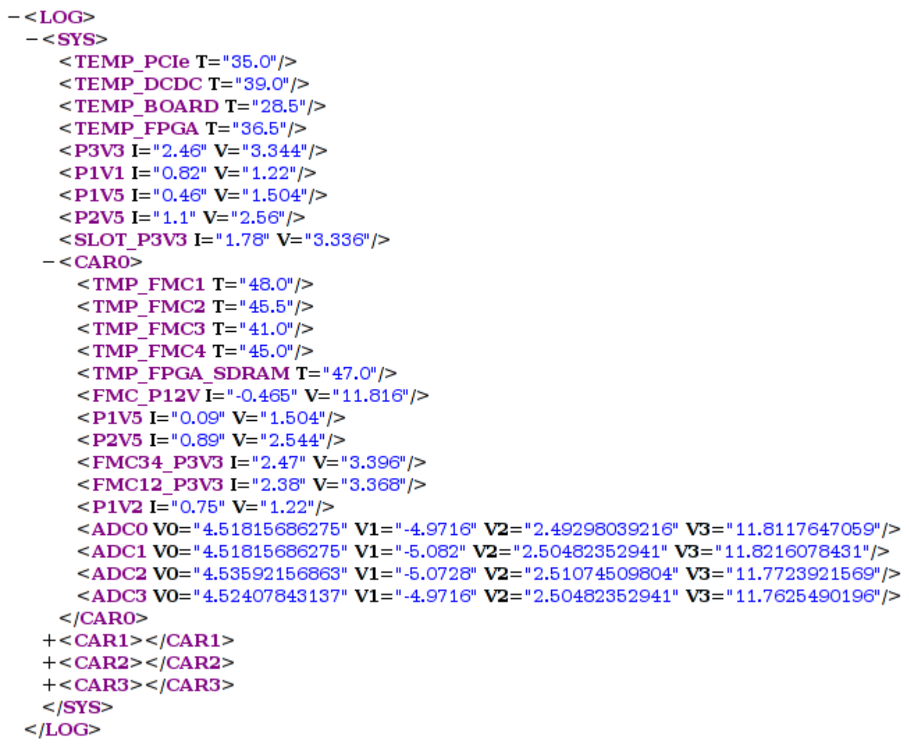
Therefore finally we have decided to use the third option:

- Sending log messages via network to the separate logserver. As our system is usually connected to the DAQ system, it should be possible to configure one of the DAQ computers to serve as the logserver for our system. In this solution the generated log messages may be quickly sent out, so even in case of system crash, we can preserve messages generated shortly before the failure. Additionally all log messages are immediately available in the DAQ server, even if our system is temporarily non-functional and requires rebooting. This method was also implemented using the Python *logging*[17] package. However the suggested implementation based on SocketHandler and LogRecordStreamHandler classes[18] relies on sending of Python objects via "pickle" protocol. Receiving such objects via the logserver creates serious security risk,[19] and therefore alternative solution, based on msgpack library was also developed.

10. RESULTS

The controller has been tested during development of the GEM based KX1 diagnostics for the JET facility.[11,20] In this system the controller services two independent GEM detectors, dedicated for measurement of tungsten and nickel X-ray spectra. Each detector contains 256 anodes (strips) connected to analog input channels. The photons from collimated, measured X-ray radiation are reflected at different angles by a Bragg crystal, so that the frequency of hits at different strips is proportional to intensity of X-ray radiation in corresponding energy range. The system measures plasma radiation spectrum as a function of time, with resolution up to 10 ms.

The low level access from Matlab was successfully used for development of control and data processing algorithms.[12] Functionality associated with the autonomous operation of the controller with external calibration loop provided by the Matlab based data processing client is currently tested. However promising results have been obtained. Example spectrogram of the plasma, measured during the test in the JET facility, by the GEM detector with readout system fully controlled by the presented controller, is shown in the Figure 6.



```
-<LOG>
  -<SYS>
     <TEMP_PCIe T="35.0"/>
     <TEMP_DCDC T="39.0"/>
     <TEMP_BOARD T="28.5"/>
     <TEMP_FPGA T="36.5"/>
     <P3V3 I="2.46" V="3.344"/>
     <P1V1 I="0.82" V="1.22"/>
     <P1V5 I="0.46" V="1.504"/>
     <P2V5 I="1.1" V="2.56"/>
     <SLOT_P3V3 I="1.78" V="3.336"/>
    -<CAR0>
       <TMP_FMC1 T="48.0"/>
       <TMP_FMC2 T="45.5"/>
       <TMP_FMC3 T="41.0"/>
       <TMP_FMC4 T="45.0"/>
       <TMP_FPGA_SDRAM T="47.0"/>
       <FMC_P12V I="-0.465" V="11.816"/>
       <P1V5 I="0.09" V="1.504"/>
       <P2V5 I="0.89" V="2.544"/>
       <FMC34_P3V3 I="2.47" V="3.396"/>
       <FMC12_P3V3 I="2.38" V="3.368"/>
       <P1V2 I="0.75" V="1.22"/>
       <ADC0 V0="4.51815686275" V1="-4.9716" V2="2.49298039216" V3="11.8117647059"/>
       <ADC1 V0="4.51815686275" V1="-5.082" V2="2.50482352941" V3="11.8216078431"/>
       <ADC2 V0="4.53592156863" V1="-5.0728" V2="2.51074509804" V3="11.7723921569"/>
       <ADC3 V0="4.52407843137" V1="-4.9716" V2="2.50482352941" V3="11.7625490196"/>
     </CAR0>
    +<CAR1></CAR1>
    +<CAR2></CAR2>
    +<CAR3></CAR3>
    </SYS>
  </LOG>
```

Figure 7. Example of readout system monitoring results, distributed via the HTTP server as an XML file. Temperatures of particular boards, power supply voltages and currents are displayed.

Example of readout system monitoring results, distributed via the HTTP server and displayed in the operator's web browser is shown in the Figure 7.

11. CONCLUSIONS

The presented embedded controller is a versatile solution aimed on control of multichannel FPGA based readout systems. Even though some functionalities were developed specifically for GEM detector, it should be possible to easily adapt them to other types of detector and front end electronics.

The embedded controller offers high flexibility - most functionalities may be remotely controlled and modified, and different access levels are available - the WWW based for the normal operator, and SSH based for the expert.

The controller is highly resistant to random power failures. Due to the ramdisk-based operation, the risk of the filesystem corruption is reduced to minimum. Booting of the whole system from the external USB flashdisk allows to keep different configurations of the system ready for immediate use.

The controller software is able to handle multiple clients, providing appropriate arbitration between requests related to monitoring and calibration procedures running in the background and high-priority requests from the central data acquisition system associated with measurements of the investigated object.

The controller offers thorough monitoring of the state of detector and the readout system. The status information may be both included with the measurement data and sent together with other log messages to the remote logging server, ensuring, that even in the unlikely situation of the controller crash, the useful diagnostic information is preserved.

The aim of this paper is to present solutions used to develop such autonomous, remotely controlled readout controller and to submit them to discussion. We hope that presented information may be useful for others solving similar engineering problems.

REFERENCES

[1] Sauli, F., "GEM: A new concept for electron amplification in gas detectors," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **386**(2-3), 531 – 534 (1997).

[2] Kasprowicz, G., Czarski, T., Chernyshova, M., Dominik, W., Jakubowska, K., Karpinski, L., Kierzkowski, K., Pozniak, K., Rzadkiewicz, J., Scholz, M., and Zabolotny, W., "Fast ADC based multichannel acquisition system for the GEM detector," *Proceedings of SPIE - The International Society for Optical Engineering* **8454**, 84540M–84540M–8 (2012).

[3] Kasprowicz, G., Czarski, T., Chernyshova, M., Czyrkowski, H., Dabrowski, R., Dominik, W., Jakubowska, K., Karpinski, L., Kierzkowski, K., Kudla, I., Pozniak, K., Rzadkiewicz, J., Salapa, Z., Scholz, M., and Zabolotny, W., "Readout electronics for the GEM detector," *Proceedings of SPIE - The International Society for Optical Engineering* **8008**, 80080J–80080J–9 (2011).

[4] Zabołotny, W., Czarski, T., Chernyshova, M., Czyrkowski, H., Dąbrowski, R., Dominik, W., Jakubowska, K., Karpiński, L., Kasprowicz, G., Kierzkowski, K., Kudła, I., Poźniak, K., Rzadkiewicz, J., Sałapa, Z., and Scholz, M., "Optimization of FPGA processing of GEM detector signal," *Proceedings of SPIE - The International Society for Optical Engineering* **8008**, 80080F–80080F–9 (2011).

[5] Pozniak, K. T., Byszuk, A., Chernyshova, M., Cieszewski, R., Czarski, T., Dominik, W., Jakubowska, K., Kasprowicz, G., Rzadkiewicz, J., Scholz, M., and Zabolotny, W., "FPGA based charge fast histogramming for GEM detector," **8903, this volume**.

[6] "Buildroot: making Embedded Linux easy ." `http://buildroot.uclibc.org/` (July 2013). [Online; accessed 18-July-2013].

[7] "The Buildroot user manual, Adding new packages to Buildroot." `http://buildroot.uclibc.org/downloads/manual/manual.html#adding-packages` (June 2013). [Online; accessed 18-July-2013].

[8] Zabołotny, W. M. and Kasprowicz, G., "Low cost usb-local bus interface for fpga based systems," *Proceedings of SPIE - The International Society for Optical Engineering* **8454**, 84540T–84540T–7 (2012).

[9] Byszuk, A., Kołodziejski, J., Kasprowicz, G., Poźniak, K., and Zabołotny, W. M., "Implementation of pci express bus communication for fpga-based data acquisition system," *Proceedings of SPIE - The International Society for Optical Engineering* **8454**, 84540N–84540N–6 (2012).

[10] Simon, F., Azmoun, B., Becker, U., Burns, L., Crary, D., Kearney, K., Keeler, G., Majka, R., Paton, K., Saini, G., Smirnov, N., Surrow, B., and Woody, C., "Development of tracking detectors with industrially produced GEM foils," *Nuclear Science, IEEE Transactions on* **54**(6), 2646–2652 (2007).

[11] Jakubowska, K., Rzadkiewicz, J., Dominik, W., Scholz, M., Zastrow, K.-D., Chernyshova, M., Czarski, T., Karpinski, L., Komarzewski, A., Czyrkowski, H., Dabrowski, R., Kudla, I., Kierzkowski, K., Salapa, Z., Blanchard, P., Tyrrell, S., Pozniak, K., Kasprowicz, G., Zabolotny, W., and JET EFDA Contributors, "Development of a 1D triple GEM X-ray detector for a high-resolution X-ray diagnostics at JET," *38th EPS Conference on Plasma Physics 2011, EPS 2011 - Europhysics Conference Abstracts* **35 1**, 716–719 (2011).

[12] Czarski, T., Chernyshova, M., Dominik, W., Jakubowska, K., Kasprowicz, G., Pozniak, K., Rzadkiewicz, J., Scholz, M., and Zabolotny, W., "Fundamental data processing for GEM detector measurement system applied for X-ray diagnostics of fusion plasmas," *Proceedings of the 40th EPS Conference on Plasma Physics* (2013). also available at `http://ocs.ciemat.es/EPS2013ABS/pdf/P5.120.pdf`.

[13] "MessagePack: It's like JSON. but fast and small. ." `http://msgpack.org/` (July 2013). [Online; accessed 18-July-2013].

[14] "ConfigParser — Configuration file parser." `http://docs.python.org/2/library/configparser.html` (June 2013). [Online; accessed 18-July-2013].

[15] Pozniak, K. T., Bartoszek, M., and Pietrusinski, M., "Internal interface for rpc muon trigger electronics at cms experiment," 269–282 (2004).

[16] Drabik, P. and Pozniak, K. T., "Maintaining complex and distributed measurement systems with component internal interface framework," *Proceedings of SPIE - The International Society for Optical Engineering* **7502**, 75022C–75022C–9 (2009).

[17] "logging — Logging facility for Python ." `http://docs.python.org/2/library/logging.html` (July 2013). [Online; accessed 18-July-2013].

[18] Sajip, V., “Logging Cookbook .” `http://docs.python.org/2/howto/logging-cookbook.html` (July 2013). [Online; accessed 18-July-2013].

[19] “pickle - Python object serialization .” `http://docs.python.org/2/library/pickle.html` (July 2013). [Online; accessed 18-July-2013].

[20] Rzadkiewicz, J., Dominik, W., Scholz, M., Chernyshova, M., Czarski, T., Czyrkowski, H., Dabrowski, R., Jakubowska, K., Karpinski, L., Kasprowicz, G., Kierzkowski, K., Pozniak, K., Salapa, Z., Zabolotny, W., Blanchard, P., Tyrrell, S., Zastrow, K.-D., and JET EFDA Contributors, “Design of T-GEM detectors for X-ray diagnostics on JET,” *Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **720**, 36–38 (2013).