ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP013

EPICS INTEGRATION FOR RAPID CONTROL PROTOTYPING HARDWARE FROM SPEEDGOAT

L. Rossa*, M. Brendike†
Helmholtz-Zentrum Berlin für Materialien und Energie GmbH, Berlin, Germany

Abstract

To exploit the full potential of fourth generation synchrotron sources new beamline instrumentation is increasingly developed with a mechatronics approach. Implementing this raises the need for Rapid Control Prototyping (RCP) and Hardware-In-the-Loop (HIL) simulations. To integrate such RCP and HIL systems into every-day beamline operation we developed an interface from a Speedgoat real-time performance machine - programmable via MATLAB Simulink - to EPICS. The interface was developed to be simple to use and flexible. The Simulink software developer uses dedicated Simulink-blocks to export model information and real-time data into structured UDP Ethernet frames. An EPICS IOC listens to the UDP frames and auto-generates a corresponding database file to fit the data-stream from the Simulink model. The EPICS IOC can run on either a beamline measurement PC or, to keep things spatially close on a mini PC (such as a Raspberry Pi) attached to the Speedgoat machine. An overview of the interface idea, architecture, and implementation; together with a simple example will be presented.

INTRODUCTION

Recently designed devices for research at synchrotron sources require complex mechanical and mechatronic designs, and therefore need advanced feedback control systems [1–3]. These control systems need to be constructed diligently and are often designed, simulated and tested with specific hardware for Rapid Control Prototyping (RCP) and Hardware-In-the-Loop (HIL) simulations. Suppliers of these hardware components often provide commercial products, or Microsoft Windows compatible, dynamic link libraries to interface their hardware. An open source interface to Linux based operating systems is often not available.

To still be able to use commercial RCP and HIL tools in the Linux based BESSY II beamline control environment another solution than the commercial ones is necessary. Figure 1 illustrations how an alternative solution can look like. The idea is to include an EPICS Input-Output-Controller (IOC) into the RCP and HIL environment and thus connect the System to the EPICS beamline control network.

Like the commercial products, the alternative solution should be also easy to use and to integrate. Therefore the following interface requirements are used:

- the RCP or HIL system should be integrable into a beamline via plug-and-play,
- the developer behind the RCP or HIL system is not an EPICS expert,
- the beamline scientist or user is not an expert in the RCP or HIL architecture,
- the full flexibility of the RCP and HIL system should be maintained,
- the developer should not struggle to keep EPICS in sync with the RCP or HIL system.

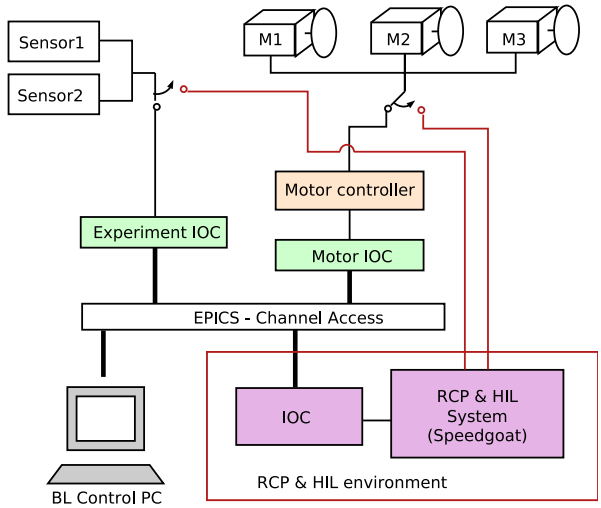

Figure 1: Integration of RCP & HIL System into beamline control environment.

As Fig. 1 indicates, in the RCP & HIL System block, hardware from Speedgoat GmbH - further referenced as Speedgoat PC - is used in this paper. However, the developed interface is compatible with hardware from other suppliers as long as MATLAB and Simulink are supported.

IMPLEMENTATION

There are two possibilities to integrate the EPICS IOC into the RCP & HIL environment. One is to integrate the IOC into the Speedgoat PC and run it directly on the real-time hardware. The second is to run the IOC on separate hardware and communicate to the Speedgoat PC via a dedicated communication interface. Despite the additional hardware requirement we chose the second option. This gives more flexibility in case the Speedgoat PC needs to be replaced and doesn't bind real-time resources for EPICS communication.

The User Datagram Protocol (UDP) is used for communication between the EPICS IOC and Speedgoat PC. This protocol was selected due to its simplicity and flexibility.

* rossa@helmholtz-berlin.de
† maxim.brendike@helmholtz-berlin.de

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP013

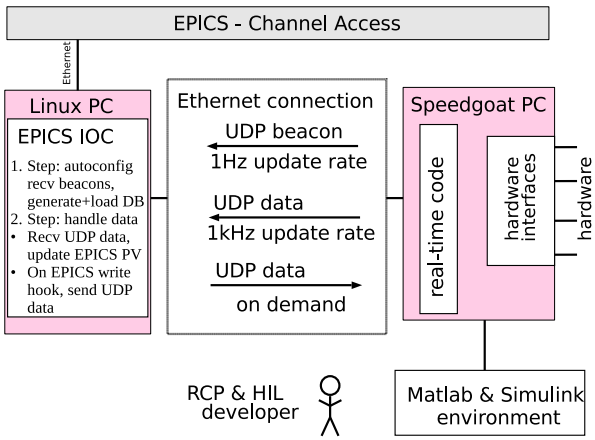

Figure 2: Communication between EPICS IOC and Speedgoat Hardware.

UDP's lacking reliability, due to the absence of handshaking between sender and receiver, is currently ignored. The Linux PC running the EPICS IOC has a dedicated Ethernet connection for a point-to-point connection with the Speedgoat PC, so package loss is unlikely. The lacking reliability of UDP is currently ignored, since the Linux PC running the EPICS IOC has a dedicated Ethernet connection for a point-to-point connection with the Speedgoat PC.

The principal data flow is pictured in Fig. 2. The Speedgoat PC sends a beacon signal every second, which includes structural information about the frequently send UDP data packages and the running real-time code. The EPICS IOC uses this beacon to auto generate an EPICS database file, and provide the corresponding EPICS records. The frequently sent UDP data packages can now be interpreted by the EPICS IOC and its data is put into the corresponding record fields.

If the EPICS user requests a change in one of the EPICS records, the EPICS IOC sends an identically structured UDP data package to the Speedgoat PC, which interprets the data and sets the corresponding variables in the real-time program.

In the following the UDP - frame architecture, MATLAB & Simulink - Library and the EPICS IOC will be discussed in more detail.

UDP - Frame Architecture

UDP is a communication protocol for transmitting arbitrary user data, wrapped up in individual UDP packages. [4] To give this user data some context, specific data frames were designed. As described earlier, the Speedgoat PC sends different signals, namely the UDP beacon and the UDP data. Both will be discussed in the following.

The UDP Beacon Signal contains information about the real-time program running on the Speedgoat PC. An example UDP beacon package is shown in Fig. 3. The package comprises a 49 byte long header with model information, followed by a stack of object information for each EPICS object that is defined in the real-time code. The object information block contains a unique object-ID, an object-type reflecting its EPICS record type, its datatype, and the EPICS PV name. In addition, information about the underlying internet protocol (IP) setup is put into each object information block, so that objects could be potentially distributed among several EPICS IOCs running on different hardware if needed.

UDP beacon
192.168.167.42 → 255.255.255.255 : 18064

	offset	name	type	example / hexdump	description
model info	0…7	timestamp (s)	*double*	7.314 0xa8 c6 4b 37 89 41 1d 40	time in seconds since program start
	8	version	*uint8*	1 0x01	always 1 to fix format
	9…48	application	*string*	sgDemo 0x73 67 44 65 6d 6f 00…	program name filled with NUL or SPACE (fixed length 40 bytes)
object info 1	49	object-id	*uint8*	1 0x01	object ID of 1st. object
	50	object-type	*uint8*	1 (AI) 0x01	other object types possible: AI,AO,DI,DO,motor,…
	51	data type	*uint8*	5 (int32) 0x05	other data types possible: (un)signed int, float, …
	52…55	ip_addr_int	*uint32*	192.168.167.42 0x2a 0xa7 0xa8 0xc0	IPv4 address of Speedgoat machine
	56…57	udp_port_int	*uint16*	18065 0x91 0x46	Speedgoat UDP port
	58…61	ip_addr_ext	*uint32*	192.168.167.41 0x29 0xa7 0xa8 0xc0	IPv4 address of EPICS-PC
	62…63	udp_port_ext	*uint16*	18065 0x91 0x46	EPICS-PC UDP port
	64…103	PV name	*string*	EPICS_ai1 0x45 50 49 43 53 5f 61 69 31 00…	name of the EPICS PV (fixed length 40 bytes)
object info 2	104	object-id	*uint8*	7 0x07	object ID of 2nd. Object
	105	object-type	*uint8*	2 (AO) 0x02	other object types possible
	106	data type	*uint8*	5 (int32) 0x05	other data types possible
	107…110	ip_addr_int	*uint32*	192.168.167.42 0x2a 0xa7 0xa8 0xc0	IPv4 address of Speedgoat machine
	111…112	udp_port_int	*uint16*	18065 0x91 0x46	Speedgoat UDP port
	113…116	ip_addr_ext	*uint32*	192.168.167.41 0x29 0xa7 0xa8 0xc0	IPv4 address of EPICS-PC
	117…118	udp_port_ext	*uint16*	18065 0x91 0x46	EPICS-PC UDP port
	119…158	PV name	*string*	EPICS_ao3 0x45 50 49 43 53 5f 61 6f 33 00…	name of the EPICS PV (fixed length 40 bytes)
end of info	159	object-id	*uint8*	0 0x00	means no more objects (normally object ID of 3rd. Object)
	160…999	padding	*uint8*	0 0x00	

Figure 3: Example of UDP beacon packets.

The UDP Data Package contains information about the current state of the EPICS objects. An example for two UDP dataframes is shown in Fig. 4. The package content starts with an 8 byte timestamp and additional meta data if used. Next follows a stack of object information frames. These object information frames contain a lot of redundant data from the beacon, like the EPICS PV name, the datatype, and the object-type; but in addition to them it includes the fields *value-count* and *value*. The field *value-count* indicates the dimension of the *value* field. If *value-count* is X, *value* contains X measures stacked in this UDP data package. The redundant object information are currently kept for debugging. For performance reasons they could be dumped in a future implementation.

Sending UDP data packages from the EPICS IOC to the Speedgoat PC, to change values on the Speedgoat PC in real-time, works identically. Here the EPICS IOC places the new value into the UDP data package's value field and the Speedgoat PC extracts the value and links it with the corresponding internal variables.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP013

UDP data analog input (AI)

192.168.167.42 → 192.168.167.41 : 18065

	offset	name	type	example	hexdump
meta data	0…7	timestamp (s) optional meta data	*double*	12.345	0x71 3d 0a d7 a3 b0 28 40
object info 1	8	object-id	*uint8*	1	0x01
	9	object-type	*uint8*	1 (AI)	0x01
	10	data type	*uint8*	5 (int32)	0x05
	11…12	value-count	*uint16*	1	0x01 0x00
	13…52	PV name	*string*	EPICS_ai1	0x45 50 49 43 53 5f 61 69 31 00…
	53…56	value	*int32*	42	0x2a 0x00 0x00 0x00
obj.2	57	object-id	*uint8*	0	0x00 (no more data)
	58…999	padding	*uint8*	0	0x00

UDP data analog output (AO)

192.168.167.41 → 192.168.167.42 : 18066

	offset	name	type	example	hexdump
meta data	0…7	timestamp (s) optional meta data	*double*	13.456	0xb6 f3 fd d4 78 e9 2a 40
object info 1	8	object-id	*uint8*	7	0x07
	9	object-type	*uint8*	2 (AO)	0x02
	10	data type	*uint8*	5 (int32)	0x05
	11…12	value-count	*uint16*	1	0x01 0x00
	13…52	PV name	*string*	EPICS_ao3	0x45 50 49 43 53 5f 61 6f 33 00…
	53…56	value	*int32*	42000	0xa4 0x10 0x00 0x00
obj.2	57	object-id	*uint8*	0	0x00 (no more data)
	58…999	padding	*uint8*	0	0x00

Figure 4: Examples of exchanged UDP data packets.

MATLAB & Simulink - Library

To hide the complexity of the UDP communication and data handling inside the Simulink code, a library with user defined Simulink blocks (Fig. 5) was created.

The library contains an *EPICS send* and *EPICS receive* block (Fig. 5 left top and left center) to give the developer access to Ethernet configuration. The use of these two blocks is mandatory for the communication to the EPICS IOC.

When the two configuration blocks are included in the Simulink code, the developer only needs to integrate the EPICS record representing blocks and wire them to the desired Simulink signals. The developer needs to provide each of the blocks with a unique object ID, the EPICS PV name

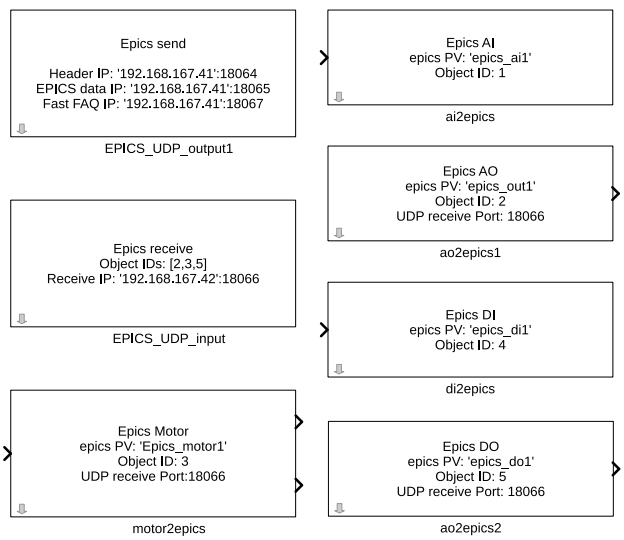

Figure 5: Simulink library for EPICS communication.

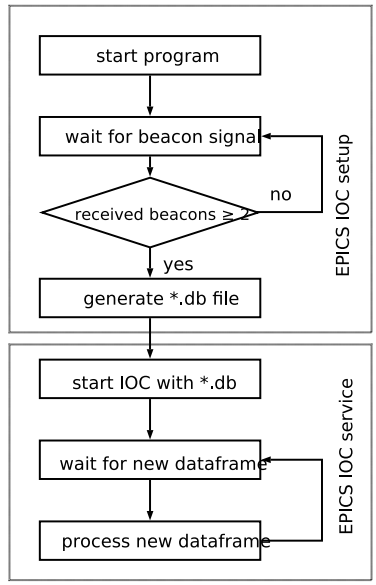

Figure 6: State flow diagram of EPICS IOC.

```
# object-id 2, data-type 5/int32, object-type 20/motor-long
record(motor, "SPEEDGOAT:Epics_motor1") {
..field(DTYP, "asynMotor")
..field(OUT, "@asyn(IP,1,1)")
...
}

# object-id 1, data-type 5/int32, object-type 40/encoder
record(longin, "SPEEDGOAT:epics_enc_pos") {
..field(DTYP, "asynInt32")
..field(INP, "@asyn(IP,0,1)SG_epics_enc_pos")
...
}
```

Figure 7: Generated DB for EPICS IOC.

and a datatype. The rest is done automatically by the EPICS IOC.

EPICS IOC

The EPICS IOC works in two steps (Fig. 6). First it receives beacon packets (Fig. 3) and generates an EPICS database file with mappings to internal code (Fig. 7). To achieve this, the EPICS IOC configures itself at startup with the received UDP beacon information. The EPICS IOC supports analog and binary inputs and outputs, the EPICS motor record and almost any field inside an EPICS PV, while using common EPICS data types[1]. It is possible to use the power of EPICS inside this IOC or elsewhere in the EPICS network.

In the second step it synchronises the generated EPICS PVs with the Speedgoat PC. UDP data packets (Fig. 4) going from the Speedgoat PC to EPICS are handled as input and are written to PVs. The EPICS IOC is based on EPICS asyn and uses it where possible. If a field of a record incompatible with EPICS asyn should be changed, EPICS base is used instead [5, 6].

Data send from the EPICS IOC to the Speedgoat PC is handled as output. Changes of EPICS PVs are detected with an internal hook mechanism and the IOC sends UDP data packets to the Speedgoat PC.

[1] integers 1/8/16/32/64 bit, IEEE754 floating point 16/32/64 bit, strings up to 40 ASCII characters

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP013

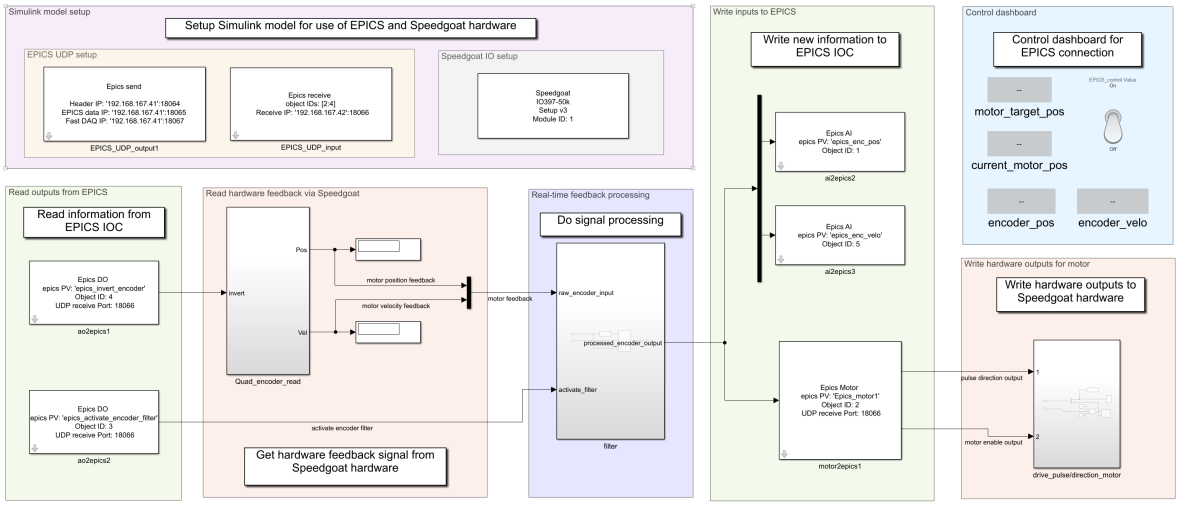

Figure 8: Simulink example model for RCP on a single stepper motor.

Figure 9: Mechanical setup for example setup.

EXAMPLE

To demonstrate the usage of the developed tools a small example will be discussed. In this example setup a signal filter for a motor feedback encoder is designed using Speedgoat real-time hardware and MATLAB Simulink.

Figure 9 shows the mechanical setup for the example. It comprises a 200 steps per revolution stepper motor and a 4000 increments per revolution quadrature encoder. Both are connected to the Speedgoat PC.

Figure 8 shows the Simulink model designed by the RCP developer. It exports the encoder feedback signals and the motor signals into EPICS PVs (second column of blocks from the right). The EPICS user can change the encoder's count direction and activate or deactivate a signal filter for the encoder feedback. This is done by the two blocks on the bottom left.

Compiling and running the program on the Speedgoat PC will automatically generate the UDP data stream. The Ethernet port configured in the EPICS send and receive blocks is used for sending the UDP packages (see the top left of Fig. 8).

Now a Linux PC running the EPICS IOC needs to be connected to the configured Ethernet port of the Speedgoat PC. Immediately after connecting the PCs, the IOC listens for the UDP beacon, automatically generate a database file and starts the actual IOC. Afterwards any EPICS user in the network can access the EPICS PVs from the Speedgoat PC via channel access.

CONCLUSION

The developed interface from Speedgoat hardware to EPICS meets the initial requirements. The RCP & HIL environment can be easily integrated via the EPICS IOC into an existing EPICS environment. The EPICS users do not need detailed knowledge about the internal structure of the RCP & HIL hardware but can access the exposed information via EPICS. Similarly the RCP & HIL developer doesn't need to know about the EPICS environment and can simply expose data from the Simulink code via the developed Simulink library.

The presented example works as a proof of the principle. It demonstrates the use of the developed tools and shows that the interface works.

A full performance benchmark still needs to be done. Next steps include a full integration of a bigger system than the example setup.

ACKNOWLEDGEMENTS

Special thanks to Roland Fleischhauer, Olaf Pawlizki, Rayk Horn, Markus Neu and David Kraft for helping with the hardware integration of the Speedgoat PC into the BESSY II electronics environment.

Thanks to the EPICS [7] community.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP013

REFERENCES

[1] R. R. Geraldes *et al.*, "Mechatronics Concepts for the New High-Dynamics DCM for Sirius", in *Proc. MEDSI'16*, Barcelona, Spain, Sep. 2016, pp. 44–47. doi:10.18429/JACoW-MEDSI2016-MOPE19

[2] M. Brendike *et al.*, "ESRF-Double Crystal Monochromator Prototype - Control Concept", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 776–780. doi:10.18429/JACoW-ICALEPCS2019-TUCPL05

[3] T. Dehaeze, J. Bonnefoy, and C. G. R. L. Collette, "Mechatronics Approach for the Development of a Nano-Active-Stabilization-System", in *Proc. MEDSI'20*, Chicago, IL, USA, Jul. 2021, pp. 93–98. doi:10.18429/JACoW-MEDSI2020-TUIO02

[4] RFC 768, https://datatracker.ietf.org/doc/html/rfc768

[5] https://github.com/epics-modules/asyn

[6] https://github.com/epics-base/epics-base

[7] https://epics-controls.org/