Working example for JSimMAST

The SimpleScada application

César Cuevas Cuesta cuevasce@unican.es

José M. Drake Moyano drakej@unican.es

Patricia López Martínez lopezpa@unican.es

CTR -Computers and Real-Time Group

Electronics and Computers Department

University of Cantabria (Spain)

Working example for JSimMAST

The SimpleScada application

Contents

1 Introduction ...3
2 *SimpleScada* elements ...4
2.1 Nodes ...4
2.2 Real-time design threads ...4
2.3 Other schedulable resources ...4
2.4 Passive resources ...4
3 *SimpleScada* at work ...5
4 SimpleScada MAST2.0 model ...6

1 Introduction

In order to illustrate the usage of JSimMAST, a simple real-time system is considered and its MAST 2.0 model is provided. The system consists of a SCADA (Supervisory Control And Data Acquisition) application deployed on 4 processors and meant for the supervision of a set of 6 magnitudes. It is named *SimpleScada* and it is a real-time system because the requirement that the signals must be sampled with delays not greater than the 10% of the sampling period has been set. An overview is depicted in Figure 1, where those four nodes as well as other application resources are shown.

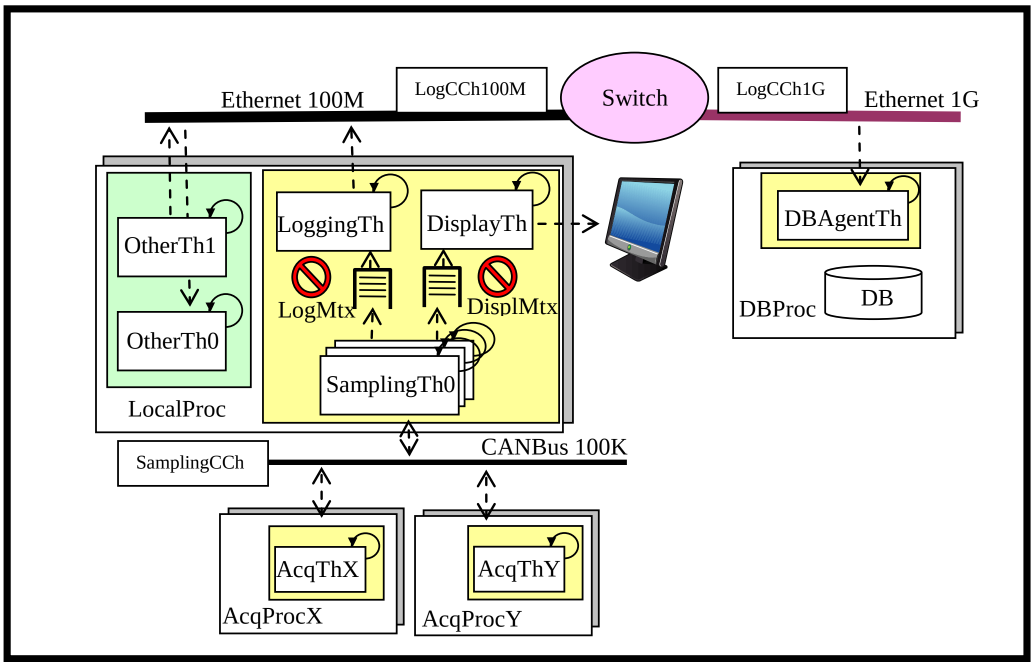


Figure 1. Overview of the *SimpleScada* application.

The description of the application elements is provided in the following section.

2 *SimpleScada* elements

2.1 Nodes

- *LocalProc*: processor where the main application elements reside, i.e. those ones that have real-time requirements. The node already has a workload represented by other applications previously deployed on it, providing the *OtherTh0* and *OtherTh1* threads.
- *AcqProc*{*X*, *Y*}: hardware devices that contain the A/D converters for reading the signals. They are accessed through a CAN Bus network of 100 KHz (*CANBus100K*) that connects them to *LocalProc*.
- *DBProc*: remote computer in which the database for the persistent record of the supervision results is placed. It is accessed from *LocalProc* through two Ethernet networks of 100MHz and 1 GHz (*Ethernet100M* and *Ethernet1G*) that are in turn interconnected through a switch (*Switch*).

2.2 Real-time design threads

Every magnitude is sampled at a certain frequency determined by configuration and every 16 signal samples, the average value is stored in the database and displayed on the screen. The application real-time design is accomplished using the following set of threads:

- *SamplingTh*{*0*, *1*, *2*, *3*, *4*, *5*}: sampling threads (one per supervised signal) whose priority is set in order to satisfy the real-time requirements.
- *LoggingTh*: recording thread that manages the persistent storage in the database, with only the throughput as requirement.
- *DisplayTh*: thread for displaying the results on the screen. It has lax real-time requirements, since at least the 95% of the records must be displayed in half the sampling time.

2.3 Other schedulable resources

- *AcqTh*{*X*, *Y*}: threads for reading the signals values.
- *DBAgentTh*: thread for the record in the database.
- *SamplingCCh*: communication channel associated to the *CANBus100K* network that allows sending messages between *LocalProc* and *AcqProc*{*X*, *Y*}.
- *LogCCh100M* and *LogCCh1G*: communication channels respectively associated to the *Ethernet100M* and *Ethernet1G* networks that allow sending messages between *LocalProc* and *DBProc*.

2.4 Passive resources

The asynchronous communication between the sampling threads and the ones for recording and displaying is performed through queues protected by the *LogMtx* y *DisplMtx* mutexes.

- *LogMtx*: Allows secure synchronization between the *SamplingThx* and the *LoggingTh* for interchange data.
- *DisplMtx*: Allows secure synchronization between the *SamplingThx* and the *DisplayTh* for interchange data.

3 *SimpleScada* at work

The functionality for reading and process data resides in the *LocalProc* and *AcqProc*{*X*, *Y*} nodes. The sampling is initiated by the *SamplingTh*{*0*, *1*, *2*, *3*, *4*, *5*} threads, which require to the *AcqTh*{*X*, *Y*} threads the task of performing de data acquisition. After that, the *SamplingTh*{*0*, *1*, *2*, *3*, *4*, *5*} threads also execute the statistical process of the sampled data, calculating for each 16 values the average one.

On the other hand, the functionality regarding information recording resides in both the *LocalProc* and *DBProc* nodes, being responsibility of the *LoggingTh* y *DBAgentTh* threads. Last, the visualization on the screen is delegated to the *DisplayTh* thread, located in the *LocalProc* node.

In order to illustrate the system activity, the sequence diagram shown in Figure 2 represents the sampling of one of the signals, e.g. the #0 signal. It is initiated by the occurrence of the timed event *SamplTrg0*, which triggers the process of sampling the #0 signal by invoking the acquisition request operation *AcqReq* on the *SamplingTh0* thread. This request is transmitted through the *SamplingCCh* communication channel by invoking on it the *SendAcqReq* operation that in turn implies invoking on the *AcqThX* thread the *ReadValue* operation for signal reading. The read value is returned through another message transmission along the same communication channel –invoking on it the *ReturnValue* operation– towards the *LocalProc* processor, where, again on the *SamplingTh0* thread, the *ProcessValue* operation is invoked. This operation is responsible of the statistical process of the sampled values, calculating the average value of each 16 values. For each calculated average value, the *GenerateLogMssg* operation is again invoked on the *SamplingTh0* thread to lay the generated value in a queue with mutually exclusive access.

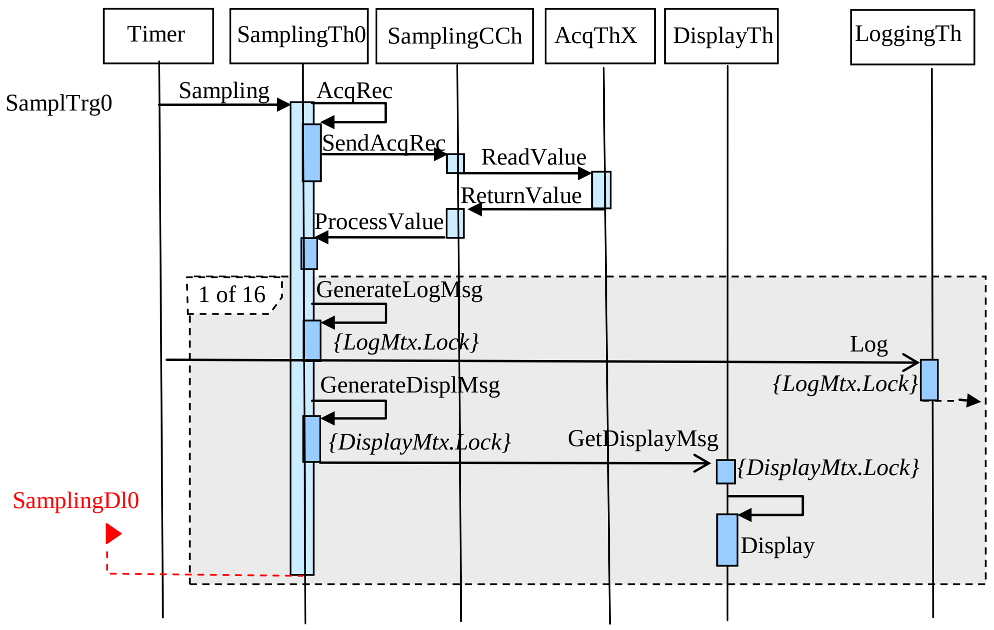


Figure 2. Sequence diagram for the *SimpleScada* application.

Every time a new average value is generated, the *Log* operation is asynchronously invoked on the *LoggingTh* thread for passing the data to be stored. Also in a mutually exclusive way, this operation accesses to the previous queue for retrieving the corresponding value, which later will be recorded in the database. Since the invocation on the *LoggingTh* thread is asynchronous, the control flow continues on the *SamplingTh0* by invoking on it the *GenerateDisplMssg* operation for the generation of the information to be displayed on the screen as well as its deposit on another queue similar to the prior one. The purpose is to invoke, asynchronously again, the *GetDisplayData* operation on the *DisplayTh* thread. Similarly to the *Log* operation, this operation for obtaining displayable data accesses to that second queue for retrieving the data that will be displayed on the screen by invoking on the same thread – *DisplayTh* – the *Display* operation.

Regarding the case of the flow control directed to the persistent record in the database, after invoking on the *LoggingTh* thread the *Log* operation, a message is transmitted through the *Ethernet100M* network by invoking on the *LogCCh100M* communication channel the *TxLoggingMssg* operation. However, since the connection between the *LocalProc* and *DBProc* nodes is not implemented directly through the *Ethernet100M* network but exists a switch that links this network with another one – *Ethernet1G* – connected to *DBProc*, it is necessary that this switch distributes the message by invoking again the *TxLoggingMssg* operation, this time on the *LogCCh1G* communication channel associated to the *Ethernet1G* network. Last, the *StoreLoggingMssg* operation is invoked on the *DBAgentTh* thread responsible for the storage in the database. Figure 3 depicts how the sequence diagram of Figure 2 continues regarding the logging control flow.

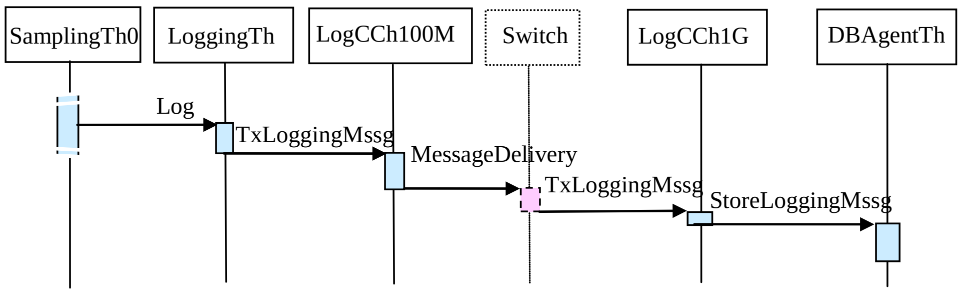


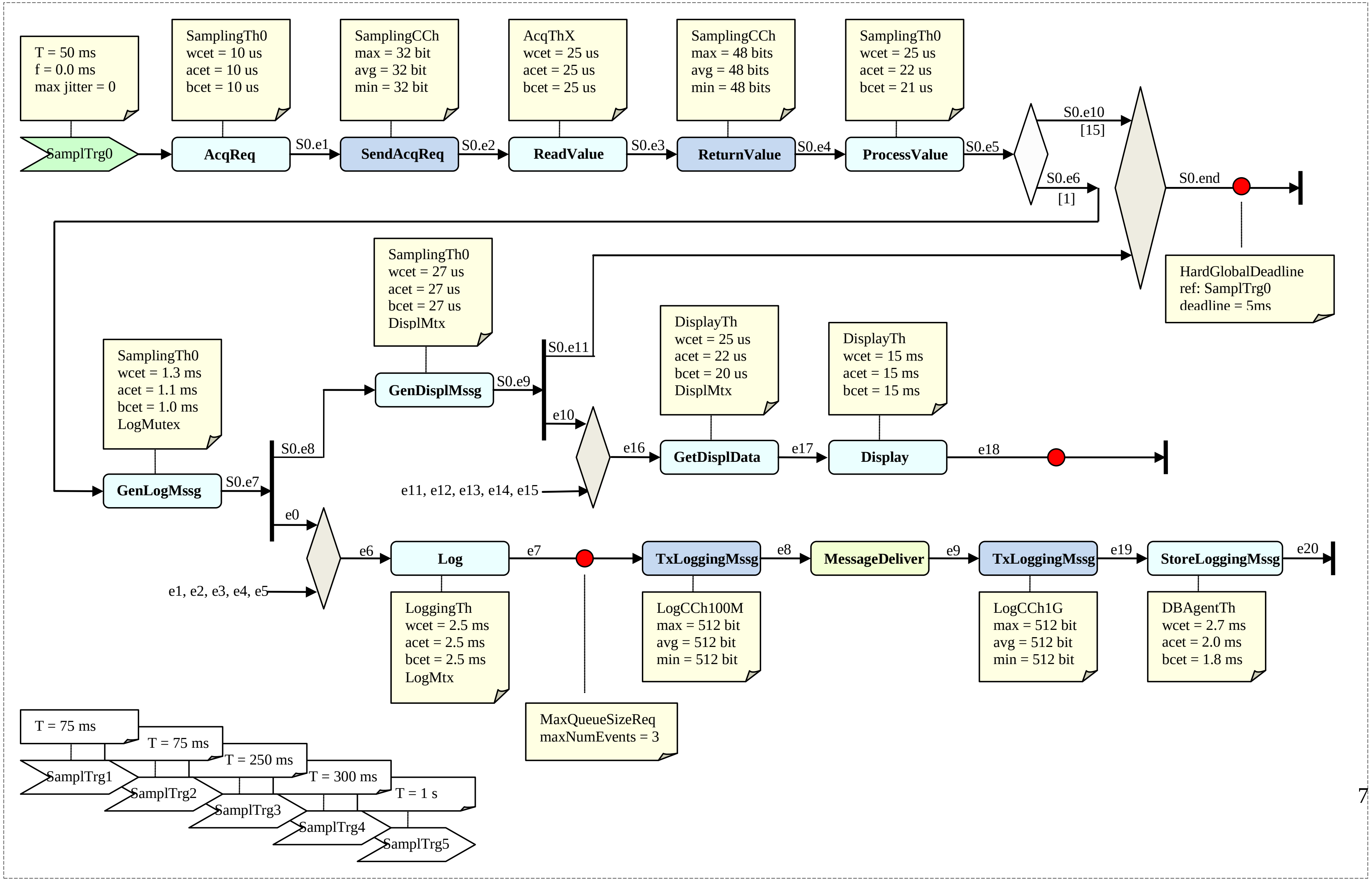
Figure 3. Sequence diagram for logging control flow

Every operation is modelled by a probabilistic variable featured by the execution times in the worst, average and best cases, with respect to a normalized processor. In addition, every operation specifies the mutexes (*LogMtx* and *DisplMtx*) that are required for its execution. For instance, the *GenerateLogMssg* operation requires locking the *LogMtx* mutex.

4 SimpleScada MAST2.0 model

The MAST model corresponding to this system is provided in XML serialized format in the *SimpleScada.mdl.xml* file. The main model data are shown in the following activity diagram.

JSimMAST working example


T = 50 ms
f = 0.0 ms
max jitter = 0
SamplTrg0
SamplingTh0
wcet = 10 us
acet = 10 us
bcet = 10 us
AcqReq
S0.e1
SamplingCCh
max = 32 bit
avg = 32 bit
min = 32 bit
SendAcqReq
S0.e2
AcqThX
wcet = 25 us
acet = 25 us
bcet = 25 us
ReadValue
S0.e3
SamplingCCh
max = 48 bits
avg = 48 bits
min = 48 bits
ReturnValue
S0.e4
SamplingTh0
wcet = 25 us
acet = 22 us
bcet = 21 us
ProcessValue
S0.e5
S0.e10
[15]
S0.e6
[1]
S0.end
HardGlobalDeadline
ref: SamplTrg0
deadline = 5ms
SamplingTh0
wcet = 27 us
acet = 27 us
bcet = 27 us
DisplMtx
GenDisplMssg
S0.e9
S0.e11
e10
DisplayTh
wcet = 25 us
acet = 22 us
bcet = 20 us
DisplMtx
DisplayTh
wcet = 15 ms
acet = 15 ms
bcet = 15 ms
e16
GetDisplData
e17
Display
e18
e11, e12, e13, e14, e15
SamplingTh0
wcet = 1.3 ms
acet = 1.1 ms
bcet = 1.0 ms
LogMutex
GenLogMssg
S0.e7
S0.e8
e0
e1, e2, e3, e4, e5
e6
Log
e7
TxLoggingMssg
e8
MessageDeliver
e9
TxLoggingMssg
e19
StoreLoggingMssg
e20
LoggingTh
wcet = 2.5 ms
acet = 2.5 ms
bcet = 2.5 ms
LogMtx
MaxQueueSizeReq
maxNumEvents = 3
LogCCh100M
max = 512 bit
avg = 512 bit
min = 512 bit
LogCCh1G
max = 512 bit
avg = 512 bit
min = 512 bit
DBAgentTh
wcet = 2.7 ms
acet = 2.0 ms
bcet = 1.8 ms
T = 75 ms
T = 75 ms
T = 250 ms
T = 300 ms
T = 1 s
SamplTrg1
SamplTrg2
SamplTrg3
SamplTrg4
SamplTrg5