ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP109

TICKIT: AN EVENT-BASED MULTI-DEVICE SIMULATION FRAMEWORK

A. Emery, G. O'Donnell, C. Forrester, T. Cobb
Diamond Light Source, Harwell, United Kingdom

Abstract

Tickit is an event-based multi-device simulation framework providing configuration and orchestration of complex simulations. It was developed at Diamond Light Source in order to overcome limitations presented to us by some of our existing hardware simulations. With the Tickit framework, simulations can be addressed using a compositional approach. It allows devices to be simulated individually while still maintaining the interconnected behaviour exhibited by their hardware counterparts by modelling the interactions between devices. Devices can be collated into larger simulated systems providing a layer of simulated hardware against which to test the full stack of Data Acquisition and Controls tools.

We aim to use this framework to extend the scope and improve the interoperability of our simulations; enabling us to further improve the testing of current systems and providing a preferential platform to assist in development of the new Acquisition and Controls tools.

MOTIVATION

Tickit's development was driven by the desire to simulate hardware triggered scans. To simulate such scans, multiple devices need to be able to communicate with one another and have linked behaviour triggered by events.

There were initial efforts to use Lewis [1], a cycle-driven hardware simulation framework for isolated devices from the European Spallation Source (ESS) and the ISIS Neutron and Muon Source. However, our requirements were not well matched to the function of this framework. This resulted in us writing bespoke solutions for our devices that grew increasingly complicated and verbose. Eventually a decision was made to develop a framework more appropriately suited to our needs.

The scans we wish to simulate require use of a Zebra [2], an event handling system utilising an FPGA from Quantum Detectors, an Eiger x-ray detector from Dectris, and a PMAC motion controller from Omron. To support scans of this nature, we would need to simulate the above devices as well as numerous motors and the communication between them. To provide this the framework needed to be:

- Multi device. To simulate a full scan we need to have many linked devices.
- Event driven. Each device in the system needs to update only when relevant, either when it changes state or when a device downstream changes an input to it.

A significant limiting factor in our hardware triggered scan simulations was the high FPGA frequency. The Zebra FPGA operates at 50 Mhz [3], a rate unachievable in a time driven system. Even with the ability to match this rate, using a time driven approach would drive the system intensely with the majority of these updates being redundant. As the simulation progresses, simulation time and real time would slowly diverge, the rate of which increasing with added complexity. By using an event driven approach instead we only need to update each device when there is a relevant change. This enables the simulation to be run at a slower overall rate, lagging behind when operations are made, but then synchronising back to real time when there are periods of no change.

DESIGN

The resulting framework consists predominantly of two parts: a scheduler, and components. Components encapsulate the simulated devices and their network interfaces, and the scheduler contains the logic to run the simulation. Components possess the operational logic for running and updating the devices they contain, and provide the interface by which the scheduler orchestrates the updating each device.

All devices possess optional inputs and outputs, which can be wired together to produce a directed acyclic graph of dependent devices. This device graphing is determined with a configuration yaml file which is used to build and run the simulation. Simple device graphing is presented in Fig. 1.

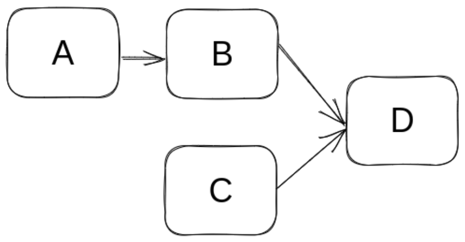

Figure 1: Device graphing. Wiring the device's inputs and outputs forms a directed acyclic graph. The scheduler ensures devices are updated in order, such that B is only updated when A has finished, and D is only updated once B and C have finished.

A summary of the framework's design and its constituent parts can be seen in Fig. 2.

The Scheduler

The scheduler orchestrates the running of the simulation. It contains references to all the components in the simulation and the wiring of all of their inputs and outputs. It is responsible for routing all the changes through the system and ensuring time is maintained.

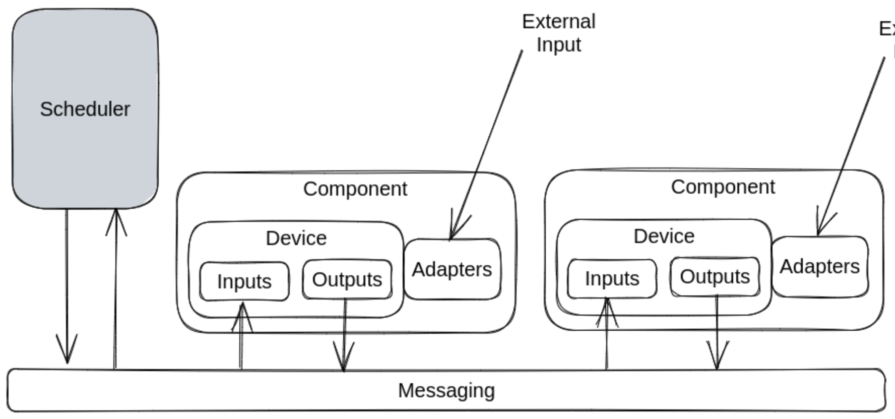

Figure 2: Summary of the Tickit design. The components host the devices and any adapters used for external communication. The scheduler keeps simulation time running and updates the components in the simulation when required.

Components

There are currently two types of components available in Tickit, device components and system components.

Device Components Are the predominant component type and encapsulate the simulated device and any corresponding adapters. This can be seen in Fig. 3. Devices in a simulation can be considered directly linked with regards to inputs and outputs.

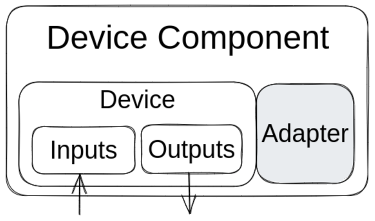

Figure 3: Device component. These components host a device and any adapters that are assigned to the device.

System Components Are a component type that nest a Tickit simulation within them. They contain a set of components, a nested scheduler for routing the updates through the components, and an optional adapter for maintaining an overview of the nested system. This is demonstrated in Fig. 4.

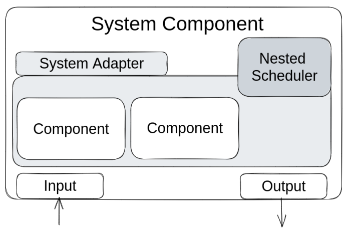

Figure 4: System components. These are comprised of a set of nested components, either device components or further system components, and a scheduler to manage them.

MAKING SIMULATIONS

Tickit simulations are configured using a yaml file containing the relevant devices and their graphing. The devices are declared with their module pathing along with any initial values for themselves or their adapters. For example, a initial amplification value for an amplifier, or a port number for a TCP adapter. The graphing is determined by the inputs field on the devices, where the outputs of other devices can be assigned. This file is used to build and run the simulation.

Devices

Devices are the user implemented code for the simulation. There is a minor amount of required boilerplate for devices; they must contain classes for inputs and outputs, and an update method. The inputs and outputs are used for wiring devices together within a simulation, and the update function contains the user implemented code for how the device should behave when it is updated.

Devices are stateful and hold the ”ground truth” of their simulated condition. When an event reaches them it triggers the update method which updates the state based on changes from both simulated inputs and adapters. This change in state can produce new output values. These new output values are acknowledged by the scheduler and cause any devices which depend on them to update in turn. The rest of the device code is typically whatever device logic is needed to support this core behaviour.

Adapters

Adapters control and read state from devices using a network interface. They were designed to provide a simple way for users to write device specific interfaces while avoiding re-implementing standard I/O logic. Each device needs a specific implementation of a given adapter which will be paired with the required I/O to allow the adapter to be run within the component.

There are currently four adapter types implemented within Tickit:

- **TCP**: An implementation that delegates the hosting of an external messaging protocol to a TCP server and utilises a regular expression pattern matching adapter for message handling. The adapter registers its methods as commands using a decorator, and attempts to match the input against a regular expression.
- **HTTP**: An implementation that delegates the hosting of HTTP requests to a server and utilises endpoints specified by a HTTP adapter. The adapter registers its methods as HTTP endpoints using a decorator.
- **ZMQ**: An adapter that publishes to a ZeroMQ [4] data stream. This is a push only adapter and does not include handling for incoming messages.
- **EPICS**: This adapter creates an EPICS [5] IOC with records associated with attributes of the device. It can be optionally initialised using an EPICS database file, and customised once this is loaded. The IOC is created using pythonSoftIOC [6], a python module developed by Diamond Light Source to enable the python interpreter to run an EPICS IOC. This adapter implementation is particularly useful when simulating devices that use hard IOC's as in these cases it is not possible to simulate the device and map it to use an instance of the real IOC.

TICKIT DEVICES

Within Diamond we have started to develop a repository of devices for the Tickit framework [7]. This currently contains a few useful devices as well as the initial Eiger and Zebra. These are undergoing development to increase their functionality.

As well as improving the devices, there is ongoing effort to deploy them within a larger end station simulation. This simulation is currently composed of numerous Lewis devices and custom simulations of specific devices, including an EIGER and a Zebra. We are working to replace these with corresponding Tickit devices.

SIMULATING AN EIGER

One of the most challenging devices implemented in Tickit so far is Dectris' Eiger detector, which is currently being adopted by several Diamond beamlines to enable a step change in data acquisition rates. Eiger has already been integrated into our high-performance detector control system, Odin [8]. This means that as long as we can keep the simulated Eiger behaviour sufficiently close to the real thing, we can deploy our Odin services (and our entire software stack on top) against it with no modifications and thus maximize the validity of our testing.

It was a significant undertaking despite only needing to implement the features that Odin and our beamlines recognise. This required us to simulate series triggering, both internal and external, and ZeroMQ streaming. The most important feature was the external series triggering, which is the enabler for simulating hardware triggered scans; the original design goal of Tickit.

Still left to develop are the event triggering modes (both internal and external), the file writing module, and the monitoring module. There is also no facility yet to customize the data the simulation produces, which is a requested feature.

SIMULATING A ZEBRA

The second key device in development is the Zebra. An initial Zebra device has been implemented utilising a system component to represent the device, and the nested device components within to represent individual logic Blocks. A depiction of the Zebra and the flow of signal through the Blocks is depicted in Fig. 5.

The initial Zebra supports AND and OR Blocks in a static, non-cyclic arrangement. Full implementation of the capabilities of a Zebra will require live rewiring; being able to change how the logic blocks are connected while the device is still running. This would require alteration of the device graph while a simulation is running. It will also require the ability for an internal Block to use the output of the Zebra as its input, potentially causing a cycle. Each of these capabilities represent a significant challenge, with the rewiring requirement needing further development of the framework's features.

The Zebra Blocks contain state which determines their output behaviours for a given input, which can be initialised with a value by the configuration file of the Zebra or set via its TCP adapter. The Zebra also contains the state of how the Blocks are arranged, and will require being able to set and read this in the expected form over TCP when live component rewiring is implemented.

Zebra blocks take 20 ns for their output signal to change in response to their inputs changing, a behaviour implemented with Tickit's tick-based time system. This delay is also implemented when changing the internal state of the block, and is accomplished by caching the new output when a change of the inputs is detected: it is hoped that a similar method will allow for the cyclic requirements of the Zebra without breaking the core assumptions of the framework.

Next steps in development of the zebra are to implement further blocks; at least DIV, GATE, PULSE, QUAD; and live rewiring.

FURTHER DEVELOPMENTS

Going forwards our efforts will be focused on developing the Zebra device and providing any framework changes required to facilitate it. The most significant of these will be the introduction of re-wirable graphs within system components needed for on the fly block manipulation within the Zebra.

Alongside this there will be efforts to extend the functionality of the Eiger, the details of which are expressed above, and to produce an initial simulated PMAC device. Preliminary work has been done to explore the requirements of a PMAC simulation however further work needs to be done to produce a working device.

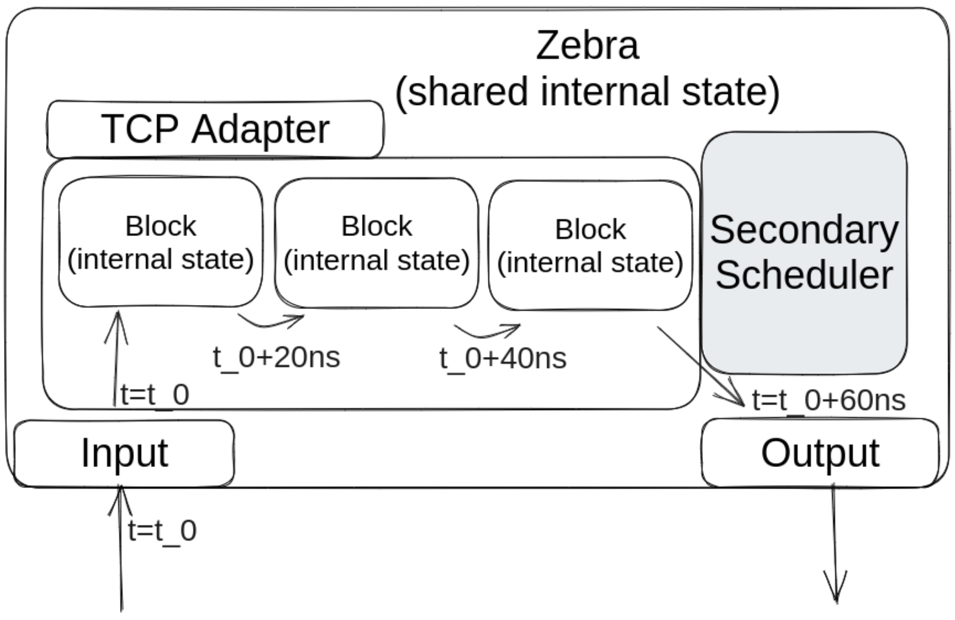

Figure 5: The structure of the Zebra device showing the flow of signal through the Blocks.

There are also goals to enable the distributed use of Tickit simulations. Tickit has been designed to allow the scheduler and components, and any subset thereof, to run in different processes. This allows us to make increasingly complex simulations by distributing the resource load. It also would allow us to create containerised deployments of sets of devices, configured for specific beamlines.

CONCLUSION

Tickit is a event driven hardware simulation framework that supports multi-device simulations. The framework and a number of devices for it have been developed to support the simulation of hardware triggered scans. We have had success integrating Tickit devices within our current simulations and use them for testing in the development of our new acquisition software stack.

The Tickit framework is in active development, with gradual framework enhancements and smaller isolated devices being created in addition to the larger goals described in **FURTHER DEVELOPMENTS**. Achieving these goals will provide us with the devices and features required to further improve and expand our beamline simulations. This in turn will provide us with greater testing capabilities and will be a valuable asset in the development, maintenance and debugging of our acquisition and controls tools.

REFERENCES

[1] D. Oram, M. Clarke, Lewis (Version 1.3.2), `https://github.com/ess-dmsc/lewis`

[2] T. M. Cobb, Y. S. Chernousko, and I. S. Uzun, "ZEBRA: a Flexible Solution for Controlling Scanning Experiments", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper TUPPC069, pp. 736–739.

[3] T. M. Cobb, I. Uzun, Y Chernousko, *Zebra Technical Manual*, Diamond Light Source, 2014, `https://github.com/dls-controls/zebra/blob/master/documentation/TDI-CTRL-TNO-042-Zebra-Manual.pdf`

[4] The ZeroMQ authors, *ZeroMQ: An open-source universal messaging library*, `https://zeromq.org/`

[5] EPICS, *Experimental Physics and Industrial Control System*, Argonne National Laboratory, `https://epics.anl.gov/`

[6] M.Abbot, PythonSoftIOC (Version 4.4.0), `https://github.com/dls-controls/pythonSoftIOC`

[7] A. Emery, tickit-devices (Version 0.3.0), `https://github.com/dls-controls/tickit-devices`

[8] T. Nicholls, odin-control (Version 1.4.0), `https://github.com/odin-detector/odin-control`