Index

1 About this White Paper

The introduced application is used as example to demonstrate the parallel task features of Zbrain SDK and explains how they can be used.

2 What's new about Zbrain SDK 4.2.0 and multiple tasks

With LabVIEW 2011 all generated code was executed serial. This reduced the application structure to a single loop.
With the update 4.2.0 for LabVIEW2013 of Zbrain SDK it is possible to run multiple tasks in parallel with different priorities. The key features are:

- Up to 5 user defined tasks per application.
- Free definition of priority per task.
- C Code generation settings per task.
- Preemptive or cooperative task scheduling.
- Communication between tasks via messages and shared memory.
- Synchronize tasks to events (interrupts).

See http://wiki.schmid-engineering.ch/zsystem4/doku.php?id=multi_tasking for more information.

3 Example Project CAN Simulator

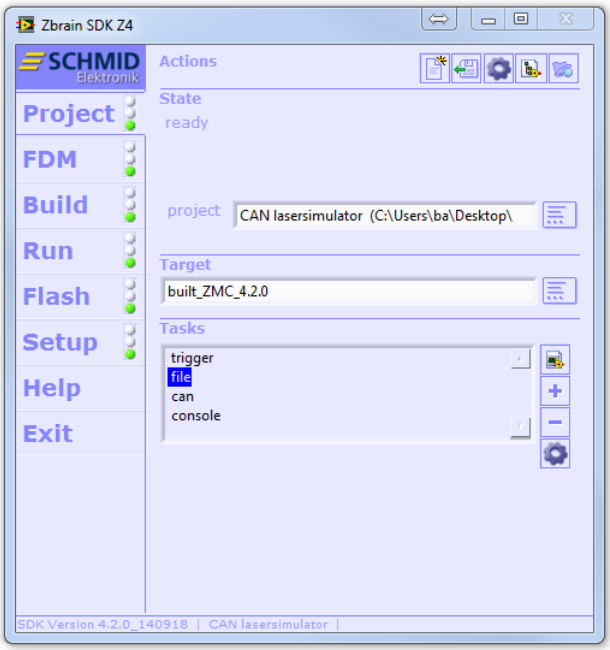
3.1 Project Topic

In multiple projects we use devices, such as 2D Laser scanners, that generate CAN streams. In order to test Hard- and Software without the need of the sometimes expensive devices and with the need to have reproducible results we decided to build a CAN stream generator based on our standard off the shelf ZMC hardware.
The simulator should generate CAN streams based on files containing the CAN frame data. Another file defines the timing details and the order, in which the streams are transmitted. The Streams should be started in a defined timing or by a trigger input with a definable delay.
The problem we face here, is that we want to provide a strict timing, while in the meantime we are forwarding CAN frames to the CAN interface and reading in the next bunch of data from file.

3.2 Software structure

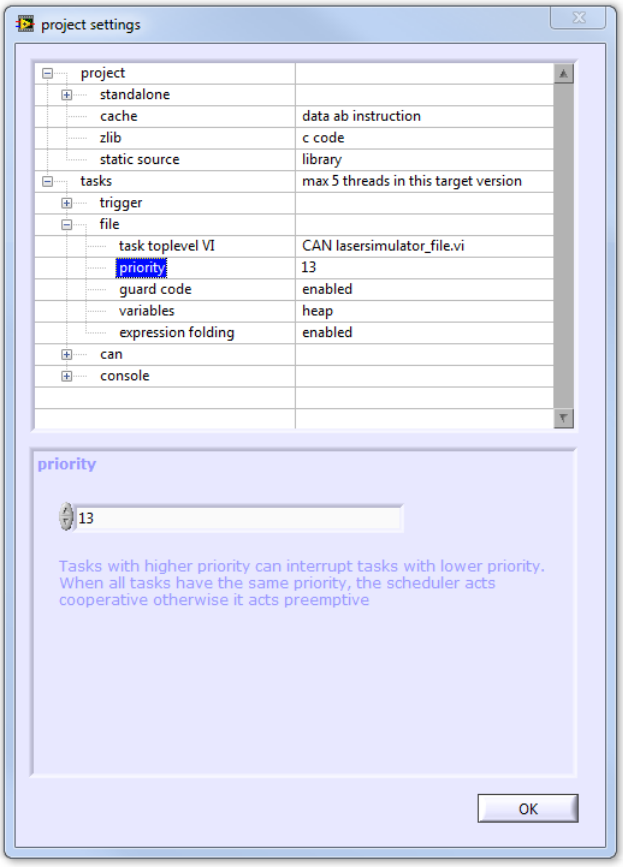
The Software is designed with the following parallel tasks:

1. CAN task:
   This task transmits the data over the CAN interface. It runs at a high priority to prevent from unwanted delay between CAN frames.
2. Trigger task:
   This task synchronizes the other tasks to achieve the desired timing. It runs at the highest priority. This task can be synchronized to a trigger signal.
3. File task:
   This task runs in background and reads in the data that are used next, whenever there is time left (when the higher prioritized tasks execute a wait VI or pend on a message or event).
4. Console Task
   This tasks receives messages from all other tasks and sends it over the UART. It runs at least priority. States, errors and debugging messages can be sent to a PC.



All tasks are represented by a top level VI and added to the project with a few clicks

Each task is individually configured:



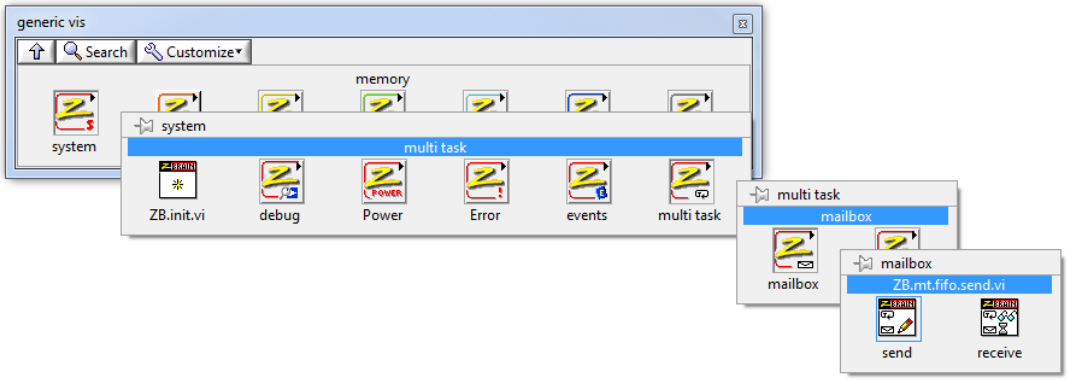
The Application is set up to use the preemptive scheduler of the underlying VDK kernel. This is done by changing the priority of the tasks.
The tasks with the highest priority, that is not in a VI with waiting function gets execution time.

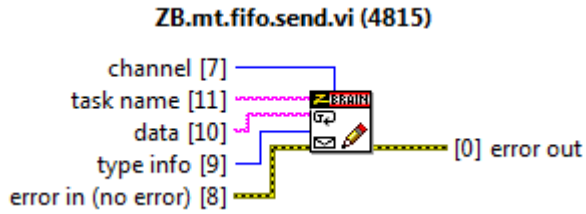
If all tasks have the same Priority the scheduler acts cooperative. Therefore, a Task switching is only done whenever a VI with wait function is executed.

3.3 Communication and synchronization

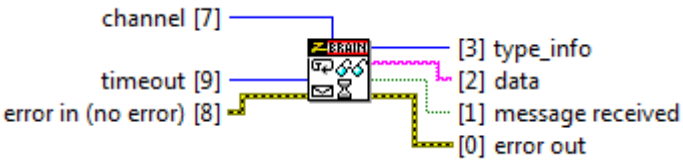
I we use Global Variables or Functional Global Variables more than on task, the code generator generates multiple variables that are independent of each other. To pass messages and share information between tasks other techniques must be used.

3.3.1 Messages





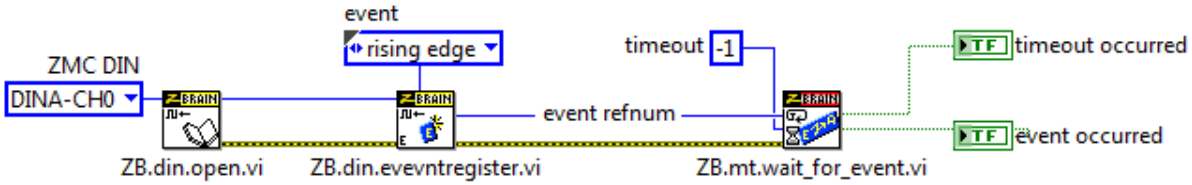
Messages are strings that can be sent to a different task. The Messages are queued in FIFO order for the receiver to read it. There is one queue for each channel and task. The receiving task is identified by its name.



The message receive VI has a timeout input that can be used to synchronize tasks. If the timeout is set to zero the VI returns immediately regardless whether a message is received or not. This can be used to poll for messages. If the timeout is set to -1 the task is halted until a message is sent from another task to this task and channel. This can be used to synchronize tasks.

3.3.2 Wait for events

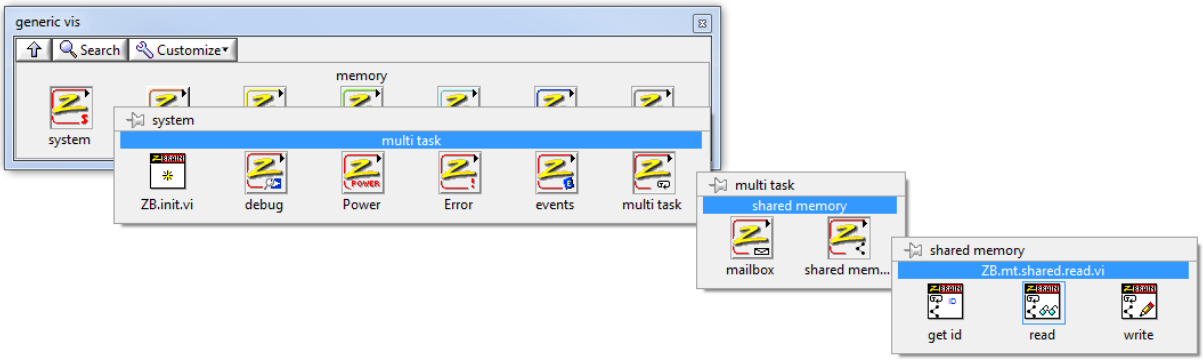
The wait for event VI can be used to synchronize a task with an event (interrupt).



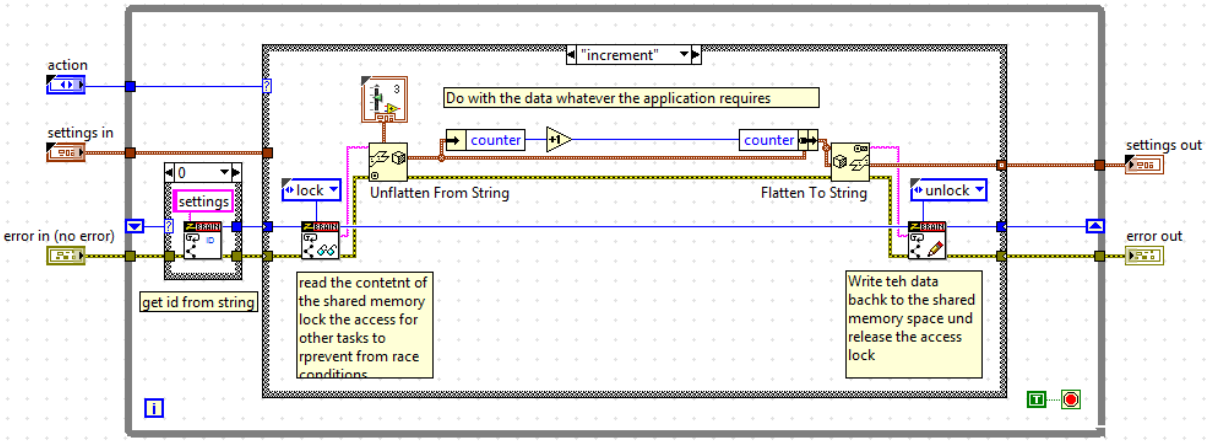
The event gets halted in this VI until the specified event or a timeout occurs.

3.3.3 Shared Memory

Sometimes the data does not have a given direction from a sending to a receiving task. It is rather altered and read by multiple tasks. A process image is an example for this. In this case, messages cannot be used to achieve these communication specifications in a convenient way.
Zbrain SDK provides Vis tat handle thread save shared memory. This provides am memory space where data can be stored and read back from any of the tasks. The multiple memory spaces are identified by strings.

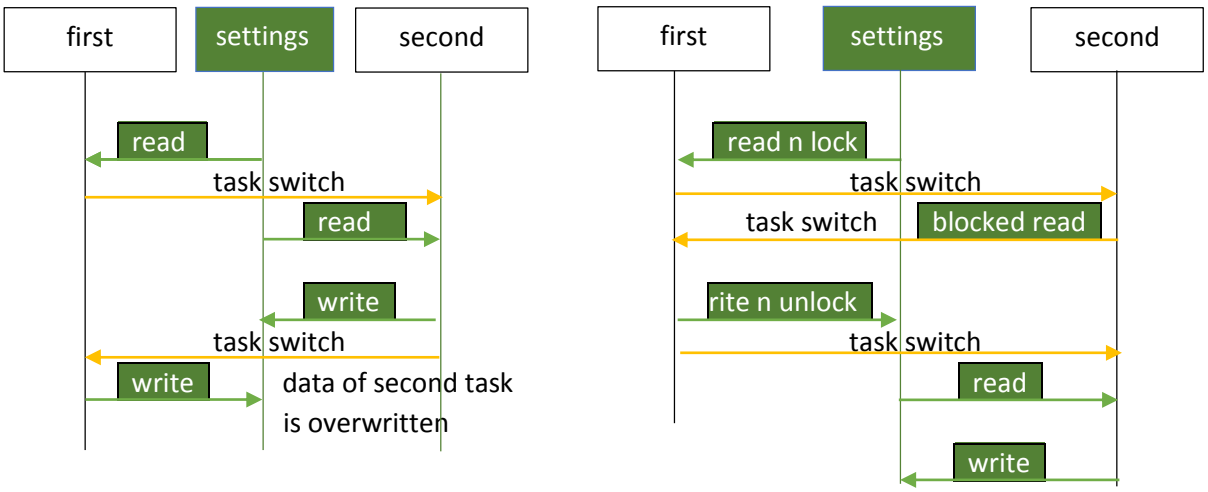


The following VI shows a multi-tasking enabled Functional Global Variable realized with the shared memory Vis:



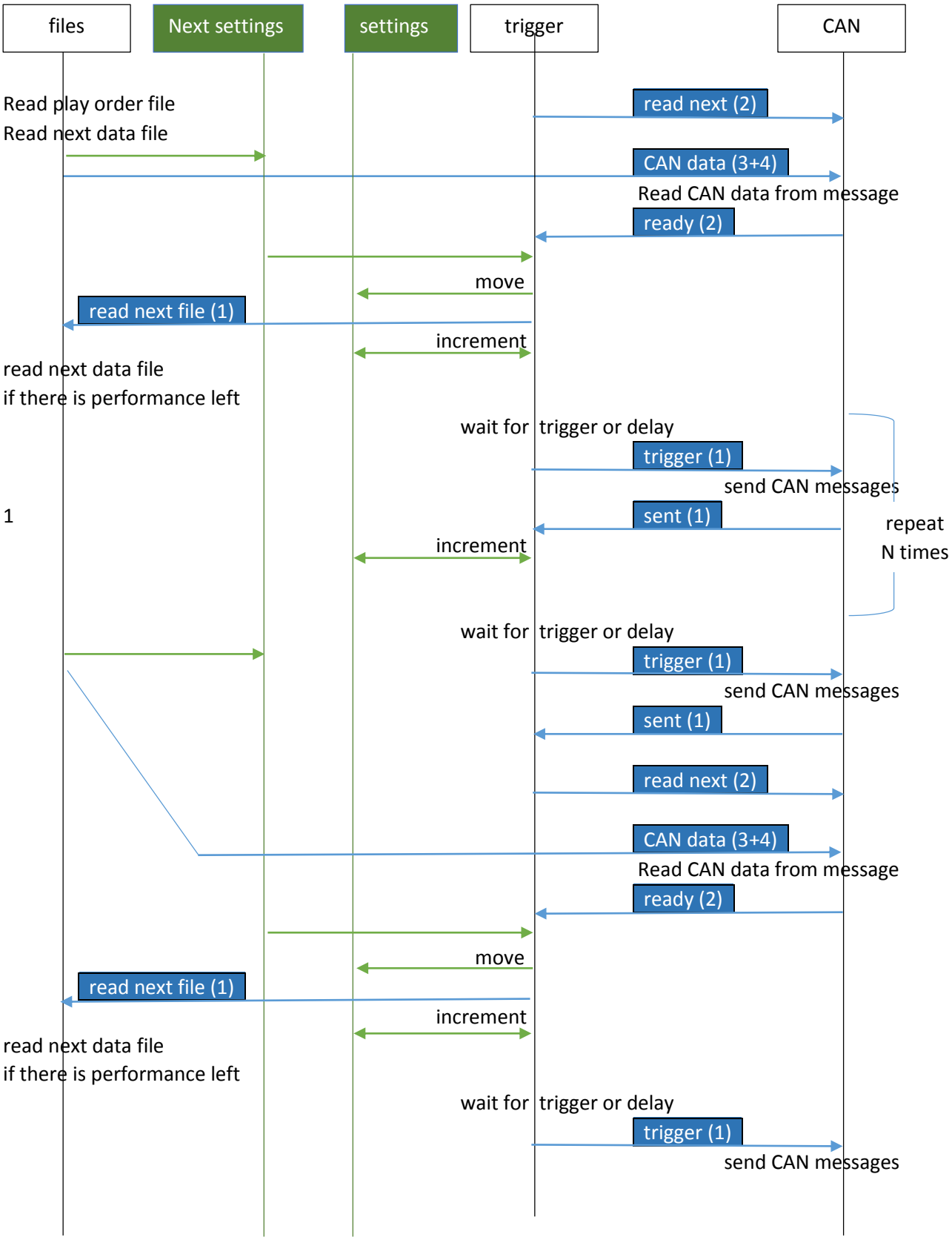
The shift register does not hold the data because it would not be synchronized between tasks. It holds just an ID that represents the shared memory space where the data reside.
Because string is the only data format, the shared memory VIs accept, the stored cluster is flattened and unflattened with the appropriate Vis. The “increment” action in the above example shows reading from and writing to a shared memory. The shared memory space is locked for other tasks when the read function is executed. When a second task would write to the shared memory after the read and before the write function of the first task is executed. The data written ba the second task would be overwritten by the write function of the first task. This is called a race condition.

The following diagram shows a race condition situation without and with locked access.
It shows tasks: name and shared memory: name.

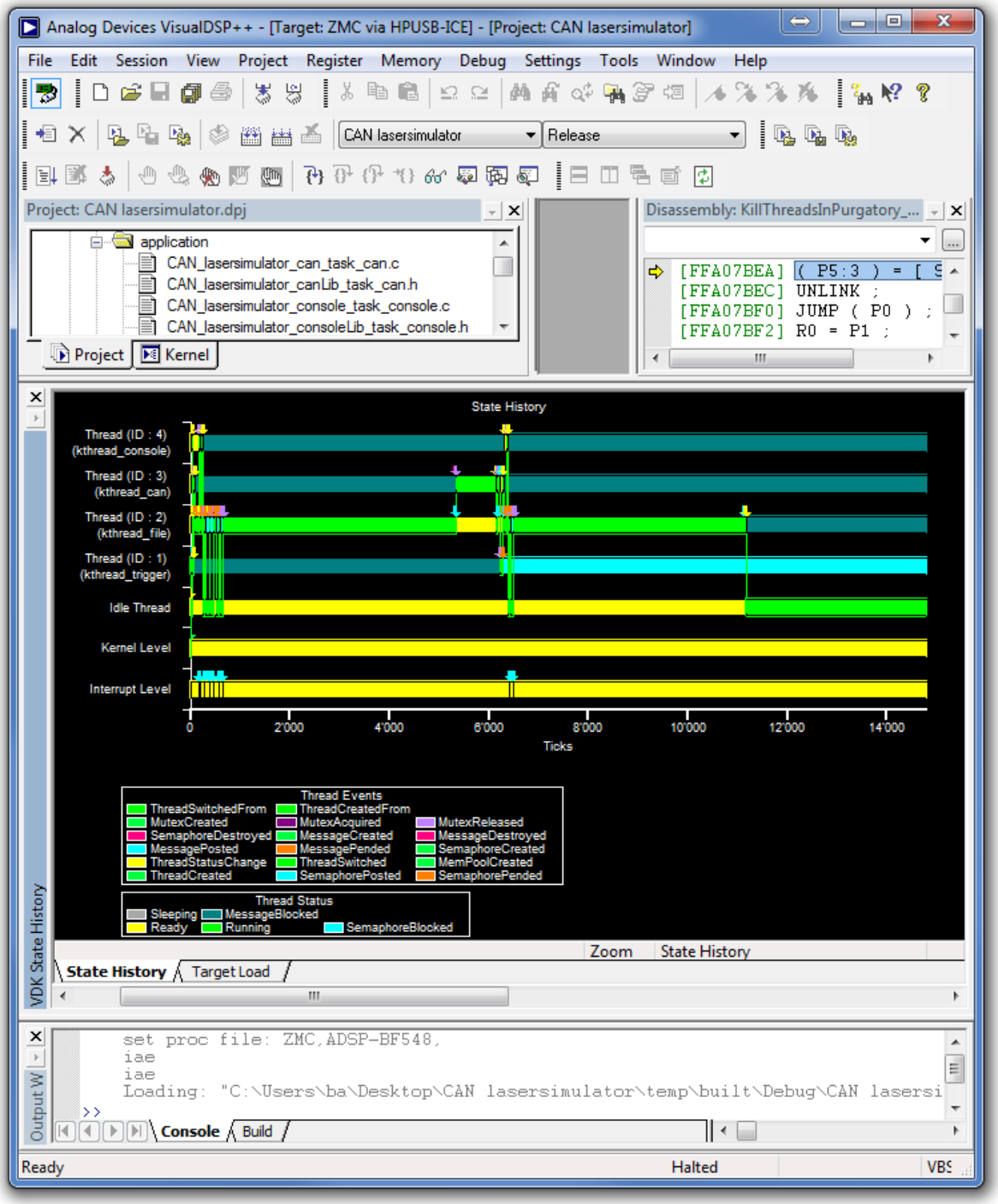


3.3.4 Communication and synchronization in the application

The following diagram shows the information that flows between the tasks.
It shows tasks: name messages: name (channel) and shared memory: name.



The interaction of the tasks van be visualized on the running application in VisualDSP++:



4 Document Revision History

Date		Revision
19.09.2014	ba	First Version
22.09.2014	ba	Wait for events added