Assignment 2
Real-Time Hardware Accelerated ROS2 Kalman Filter

Frederik Nyboe

Fall 2022

1 Description

The task is to build a ROS2 system which applies Kalman Filtering in real-time using hardware acceleration in the FPGA. A conceptual diagram of the intended system is shown in Fig. 1.

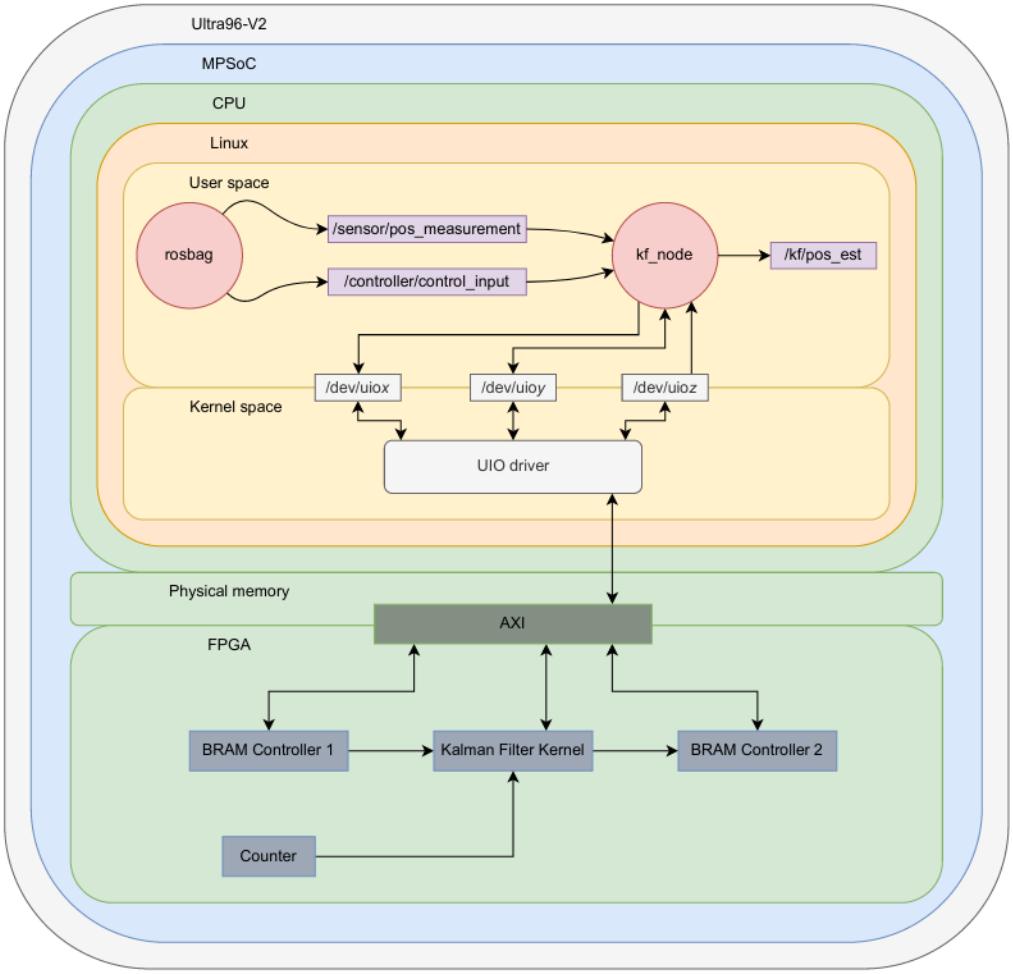

Figure 1: Conceptual diagram of the ROS2 Hardware Accelerated Kalman Filter system

The intended system estimates the cartesian position of a drone in flight. The `kf_node` node subscribes to two topics; The `/sensor/pos_measurement` topic provides noisy measurements of the drone's position with a frequency of approximately 10 Hz, and the `/controller/control_input` topic provides the control inputs applied to the drone with a frequency of approximately 10 Hz. For testing and validation, a ROS2 bag is provided which contains the a recording of the two topics from the drone in flight.

A node, `kf_node`, must be written which subscribes to the two topics. Upon having received a message from each topic (they might not always arrive in the same order), the node must write the data through `AXI` to `BRAM 1`, wait for the `Kalman Filter Kernel` to be ready, start the `Kalman Fitler Kernel` to progress the Kalman Filter a single step, wait for it to be finished, and then read the resulting position estimate from `BRAM 2`. The node should then publish the current position estimate on the `/kf/pos_est` topic.

The actual Kalman filtering must be accelerated in the FPGA using a kernel implemented in HLS. The code provided in lecture 5 can be used, however it should be rewritten to only compute a single Kalman filter step at a time. Additionally, since it might not be triggered with an exact frequency of 10 Hz, a `Counter` must be written in VHDL which supplies to the Kalman filter kernel a counter value. The difference in counter value between two consecutive steps should then be used to compute Δt, which should then be used in the Kalman filter formulation to achieve true real-time computation.

2 Development Strategy

The system must be developed in two different Git repositories; An MPSoC4Drones repository for the FPGA and OS development, and a repository for the ROS2 development.

2.1 MPSoC4Drones

The MPSoC4Drones development should be carried out as have been done in the previous lectures. Coarsely, the process is

1. Develop the Kalman filter kernel in Vitis HLS
   (a) Test in C simulation
2. Develop the counter VHDL module
   (a) Test in Vivado RTL simulation
3. Integrate the system in Vivado
4. Build the hardware (`mp4d-build -V`)
5. Test the hardware in bare-metal using Vitis
6. Put the correct entries in the device tree file
7. Build the PetaLinux project (`mp4d-build --petalinux-config -P`)
8. Connect the SD card to your computer and update the boot files, retaining your root filesystem (`mp4d-package -B`)

2.1.1 Vitis HLS

You will need to modify the Kalman Filter HLS code supplied in lecture 5. Add a port to read the counter value. In HLS, you can use the `ap_uint<>` data type with interface directive `ap_none` to accomplish this.

You also need to retain certain variable values across calls to the kernel. This can be done by declaring the variables in question as `static`.

2.1.2 Counter

The counter should be written in VHDL. The simple counter used in this assignment should take as input the clock and output the current counter value. An internal `unsigned` signal should be incremented on each rising edge of the clock. Make the width of the clock large enough to accommodate the 10 Hz system frequency plus some extra margin.

2.1.3 Vitis

In order to test the developed system in bare-metal, you can use the data provided in lecture 5 and call the Kalman filter kernel one step at a time and output the results to the screen. You can trigger delay between the kernel calls using the C `sleep()` method.

2.1.4 PetaLinux

Remember to update the device tree file by adding entries for both `BRAM` controllers and the `Kalman Filter Kernel`, in order to be able to access them from Linux with `UIO`.

2.2 ROS2

A ROS2 package skeleton is provided to get you started. The points you need to consider are the following:

- Adding the BRAM C++ driver and the `Kalman Filter Kernel` Linux driver to the package
- Writing the `kf_node` source code

In the supplied template, the `CMakeLists.txt` and `package.xml` files are already updated to facilitate the intended functionality, so you dont't need to worry about them. Additionally, the `src/` folder contains the source files for the `kf_node`, and this is where your development will take place.

2.2.1 Repository Setup

Create a new Git repository and clone it to your computer. Extract the provided ROS2 package skeleton into your clone folder. Add and commit everything.

Enter your repository and add the BRAM C++ driver as a Git submodule:

```
cd <ROS2 repository directory/src
git submodule add git@github.com:DIII-SDU-Group/BRAM-uio-driver.git
```

Upon having developed the `KalmanFilterKernel` in Vitis HLS, put the IP driver `src/` folder into the `src/KalmanFilterKernel-driver/` folder of your ROS2 repository, such that `ls src/KalmanFilterKernel-driver/src` yields

```
Makefile
xkalmanfilterkernel.c
xkalmanfilterkernel.h
xkalmanfilterkernel_hw.h
xkalmanfilterkernel_linux.c
xkalmanfilterkernel_sinit.c
```

2.2.2 ROS2 Topics

Both the `/sensor/pos_measurement` and `/controller/control_input` topics are of the type `std_msgs::msg::Float32MultiArray`. The `/kf/pos_est` topic should be of type `geometry_msgs::msg::PoseStamped`. Utility functions have been added to the `src/kf_node.cpp` and `src/kf_node.h` files for your convenience.

2.3 Enabling Networked ROS2

ROS2 automatically uses all network interfaces which a multicast enabled. That means that you can run your system on the Ultra96-V2, play back the ROS2 bag on your host machine, and visualize the estimated position using ROS2 visualization tools on your host machine in real-time. Having connected your Ultra96-V2 to your host machine using ethernet as done in lecture 4, you can enable mutlicast using the command

```
sudo ifconfig <network interface> multicast
```

Here, the network interface can be found using `ifconfig -a`. This needs to be done on both your Ultra96-V2 and on your host machine.

3 Deliverables

A report must be submitted, describing in detail the development process and design choices made. The report must include simulation and testing results for all sub steps. Links must be provided to both Git repositories. Finally, a video must be submitted showing the system running with visualization of the estimated positions in `RVIZ2`. The work should be carried out in the groups.