

Discovery Session 6

Adding control to the uP scrolling LEDs

Overview

In the previous session you programmed a processor to display the KnightRider pattern on the tricolor LEDs. What the design did not include was any means of controlling the rate of the KnightRider pattern.

In this session a custom instrument will be added, controlling a delay in the embedded code that is creating the KnightRider pattern on the LEDs.

Prerequisites

This tutorial assumes you have completed the previous tutorial, **Discovery Session 5 – scrolling the LEDs with a microprocessor**. The set of project files created in that tutorial are used as the starting point for this tutorial.

Design detail

This exercise uses the component listed in Table 1 to extend the OpenBus design originally created in Session 5, to give the OpenBus design shown in Figure 1. Note that the top schematic sheet remains the same.

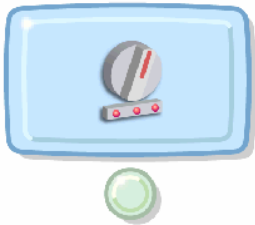
Component	Library	Name in Library
<p>CUSTOM_INSTRUMENT_1</p> 	OpenBus Palette	Custom Instrument

Table 1. List of components required by the design

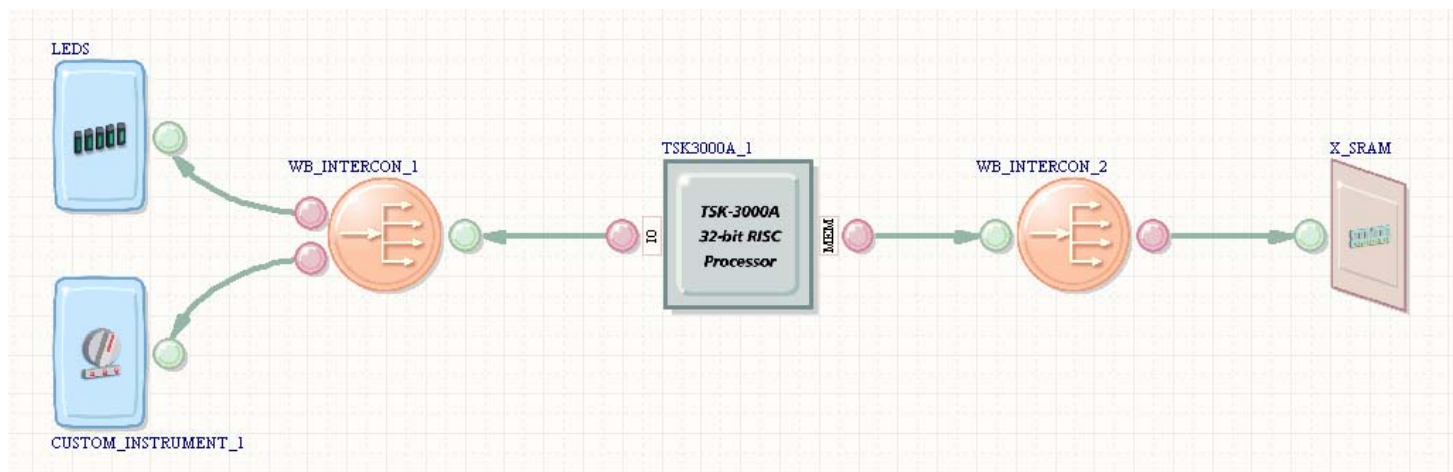





Figure 1. OpenBus document for the uP_KnightRider_wControl design.

Tutorial steps – updating the OpenBus hardware

1. Copy the contents of the Session 5 \Project folder to the Session 6 \Project folder.
2. Open the FPGA project `uP_KnightRider.PrjFpg` in the Session 6 Project folder, then right-click on the FPGA project in the Projects panel and select Save Project As, naming it as `uP_KnightRider_wControl.PrjFpg`.
3. Open the OpenBus document (`uP_KnightRider_OB.OpenBus`) in the project.
4. Open the OpenBus Palette, then locate and place the **Custom Instrument** peripheral, placing it below the LEDs OpenBus component.
5. To wire this component in, add an OpenBus **Port**  to the Interconnect on the IO side of the processor, then place a **Link**  between the new port and the new Custom Instrument. Arrange the OpenBus components as shown in Figure 1.
6. The Custom Instrument will be used to control the rate of the KnightRider pattern on the LEDs. To configure it, double-click on the Custom Instrument component to open the *Configure OpenBus Custom Instrument* dialog.
7. The dialog has 3 tabs. Bring the Properties tab forward if it is not already. This instrument will be used to output a value that sets a delay in the embedded code. To do that edit the default Output Signals entry, changing it to `Delay[7..0]`. The Initial value will be left at zero.
8. There are no inputs needed to the instrument, so remove the default **Input Signal** with .
9. Switch to the **Layout** tab to display the design of the custom instrument. Click once in the main area of the instrument face, then using the small resize handle that appears decrease the height of the instrument so that it occupies about half the height area available in the dialog, as shown in Figure 3.
10. The height will be controlled via a **TrackBar**, click to place one from the Instrument Controls section of the dialog (as shown in Figure 2), then click on the instrument face to place it. The new TrackBar can be dragged to re position it.

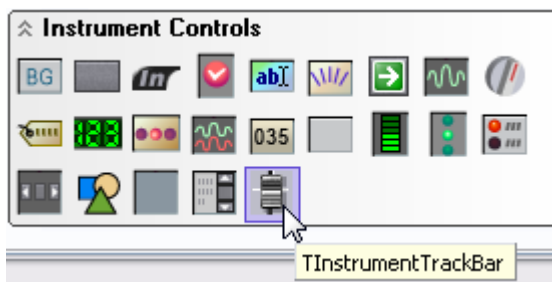


Figure 2. Place a TrackBar to control the rate of the LEDs.

11. While the TrackBar is selected, edit the **MaxValue** to be 255, the **Name** to be `FaderDelay`, and the **SignalName** to be `Delay[7..0]` as shown in Figure 3.

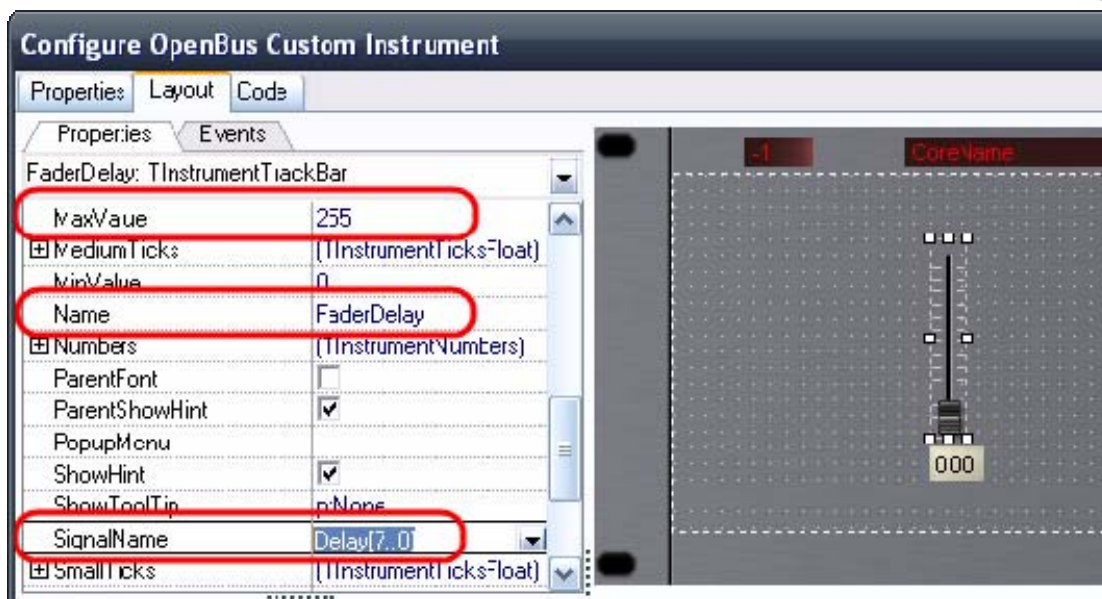


Figure 3. Place a TrackBar, and configure the MaxValue and the Name.

- To display the current setting of the TrackBar we'll add a **NumericPanel**. Click to place one from the Instrument Controls section of the dialog, as shown in Figure 4, then click on the instrument face to place it. Position it below the TrackBar.



Figure 4. The NumericPanel will display the current delay setting.

- Configure the following properties of the NumericPanel: **DigitsInGroup** to 3, **Digits** to 3, **Name** to DisplayDelay, and **Radix** to rdxDecimal.
- The next step is to put some code behind the custom instrument, code that will update the NumericPanel to show the current value for the delay slider. Select the FaderDelay slider again, switch to the **Events** sub-tab of the **Layout** tab, and double-click the **OnChange** event value box - enter the following code:

```
procedure TDesignedAreaPanel.FaderDelayChange(Sender: TObject);
begin
    DisplayDelay.Value := FaderDelay.Value;
end;
```

- Click the Accept button to save these changes you have made to the Custom Instrument.
- Compile the FPGA project and check and resolve any errors detailed in the Messages panel.
- This completes the OpenBus part of the design, save the OpenBus document.

Tutorial steps – updating the software platform

- The software platform needs to be updated, to include the new Custom Instrument. To do this, open the SwPlatform document in the Embedded project and click the **Import from FPGA Project** button. The Custom Instrument wrapper will be added.
- Click to select the new Custom Instrument wrapper, and then click the **Grow Stack Up** button to open the *Grow Stack* dialog, from where you can add the **Custom Instrument Driver**. Once you have closed the dialog, the software platform will look like Figure 5.

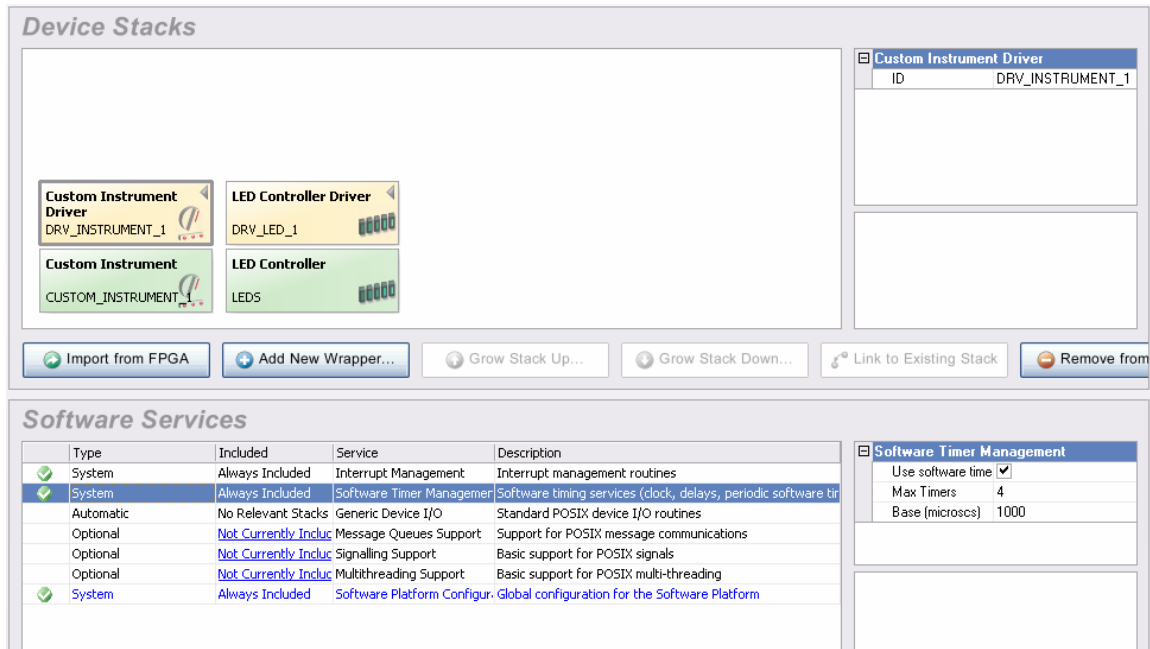


Figure 5. The I/O wrappers and low-level drivers are managed by the Software Platform.

- The software platform is now ready; the last phase is to update the embedded application to include a delay. Select **File»Save All** to save all your work.

Tutorial steps – updating the embedded code

21. With the SwPlatform document still open and the **Custom Instrument Driver** selected, hit F1 to bring up the Knowledge Center panel. Note the data type `instrument_t` which will be used for connecting our code to the custom instrument hardware, through functions such as `instrument_open`.
22. Click on the [instrument_open](#) link to see the implementation details, noting that we will also need to use `#include <drv_instrument.h>` in our code. Close the Knowledge Center panel.
23. Open the existing `main.c` document from the `uP_KnightRider` embedded project. Near the top, under the existing `#includes`, add the following to include the driver and related information:

```
#include <drv_instrument.h>
#include "instruments.h"
```

24. Further down below the existing function prototypes, add the following variables:

```
instrument_t * ctrl_spd;
unsigned char inst_delay = 255;
```

25. Add the following line of code to the `void init(void)`, after the `ptrLEDS = led_open(LEDS);` code:

```
ctrl_spd = instrument_open(CUSTOM_INSTRUMENT_1);
```

26. Finally, modify the main program loop as depicted, the before and after are shown below:

```
while(1)
{
    if (Tick)
    {
        UpdateKnightRiderLEDS();
        Tick = 0;
    }
}
```

↓

```
while (1)
{
    if (Tick)
    {
        if (inst_delay++ == 255)
        {
            /*
             * Read Custom Instrument Slider Value and subtract it from
             * delay time (higher slider = less delay = faster chasing).
             */
            inst_delay = instrument_get_value( ctrl_spd, CUSTOM_INSTRUMENT_1_DELAY);
            UpdateKnightRiderLEDS();
        }
        Tick = 0;
    }
}
```

27. Save your work using **File»Save All** and open the **Devices View**.
28. Build and download your updated project – if you get any errors during compilation you will need to go back and double-check your work, ensuring all fields, names and code have been entered correctly.
29. Open the Custom Instrument – adjust the slider upwards to speed up the rate of the LED scanning pattern.
30. To improve the resolution of the slider, reduce the periodicity of the timer interrupt down to 1000 microseconds, the before and after are shown below:

```
void init(void)
{
  ...
  timer_register_handler(0, 2000L, (void*)TimerTick);
}
```



```
void init(void)
{
  ...
  timer_register_handler(0, 1000L, (void*)TimerTick);
}
```

Some additional items for you to try

Below are some suggestions for additional things you might try as you further explore the tools and the new environment. These changes are not required by later exercises and it's recommended you save a backup of your current source files first:

- Improve the speed control algorithm so that the adjustment is perceptually linear rather than exponential.
- Extend the design to have Red, Green and Blue LED sets scanning separately.
- Extend the instrument to have three sliders for Red, Green and Blue LED scan speeds.

Revision History

Date	Revision No.	Changes
29-Jul-2009	1.0	New document release

Software, hardware, documentation and related materials:

Copyright © 2009 Altium Limited.

All rights reserved. You are permitted to print this document provided that (1) the use of such is for personal use only and will not be copied or posted on any network computer or broadcast in any media, and (2) no modifications of the document is made. Unauthorized duplication, in whole or part, of this document by any means, mechanical or electronic, including translation into another language, except for brief excerpts in published reviews, is prohibited without the express written permission of Altium Limited. Unauthorized duplication of this work may also be prohibited by local statute. Violators may be subject to both criminal and civil penalties, including fines and/or imprisonment. Altium, Altium Designer, Board Insight, Design Explorer, DXP, LiveDesign, NanoBoard, NanoTalk, P-CAD, SimCode, Situs, TASKING, and Topological Autorouting and their respective logos are trademarks or registered trademarks of Altium Limited or its subsidiaries. All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed.