

## Summary

Tutorial

TU0131 (v2.0) March 20, 2008

This tutorial is based on the TRAININGcenter Video of the same title and provides a step by step overview of how to create a reasonably sophisticated FPGA design. It outlines how to create a design using OpenBus, Constraining the Design to a target device, developing embedded software using DSF, and running the entire design on a Desktop NanoBoard.

FPGAs have been around for over 20 years but for many designers they are still a 'new thing'. Coming to grips with the technology and their capabilities can sometimes feel like a daunting task but regardless of your current skill levels, this tutorial will walk you through the process of creating a reasonably sophisticated FPGA design from start to finish. In doing so, you'll be provided with an overview of several of the features of Altium Designer that have been specifically developed to increase productivity and to make the design process easier. These include:

- OpenBus system development
- Autoconfiguration of FPGA projects
- Device Software Framework

You'll also be shown how to interact with some of the advanced Desktop NanoBoard peripherals such as:

- Composite video capture
- Touchscreen TFT display

## What you'll be creating

This tutorial is based around a video capture and display application.



Figure 1. Overview of the Video capture system

## What you'll need

In order to complete this tutorial, you'll need:

- Altium Designer 6.8 (or later) installed
- A Desktop NanoBoard with PB01 peripheral board and DB30 Spartan3 daughter board (or similar) installed.
- A composite video source from a camera or DVD player.
- Vendor build tools. The free version of Xilinx's ISE (8.2.03i or later) will be sufficient if you are using the DB30 daughter board.

## Creating the FPGA Project

To start working with Altium Designer, you first need a project. A project makes managing your source design documents and any generated outputs much easier. For FPGA designs, you'll need an FPGA project.

To create a new FPGA project:

1. Select **File » New » Project » FPGA Project** from the menus, or click on **Blank Project (FPGA)** in the **New** section of the **Files** panel.
2. The **Projects** panel will display a new FPGA project with the default name **FPGA\_Project1.PrfPpg**. Select **File » Save Project** or right-click the project in the **Projects** panel and select the **Save Project** item. Save the file as **SpinningVideo.PrfPpg**.

Warning: Do not use spaces or dashes ('-') in file or project names. Use underscores if necessary.

To avoid problems that some of the FPGA build tools have with spaces (' ') in filenames, use underscores ('\_') instead of spaces.

## Adding source documents to the FPGA Project

An FPGA project supports three types of source documents – Schematic, HDL (Verilog or VHDL) and OpenBus. You can use a mixture of all three document types in a project with the use of sheet symbols. However, for FPGA projects, you *must* have a schematic as the top level document. This is necessary for supporting FPGA-to-PCB integration and synchronization.

To create a single schematic document and add it to the project:

1. Select **File » New » Schematic**, or click on **Schematic Sheet** in the **New** section of the **Files** panel. A blank schematic sheet named **Sheet1.SchDoc** displays in the design window
2. Rename the new schematic file (with a **.SchDoc** extension) by selecting **File » Save As**. Navigate to the same folder as your project and type the name **SpinningVideo\_FPGA.SchDoc** and click on **Save**.

We'll come back to this document shortly but for now, we need to also create a new OpenBus document and add that to the project as well.

To create a single OpenBus document and add it to the project:

1. Select **File » New » Other » OpenBus System Document**, or click on **OpenBus System Document** in the **New** section of the **Files** panel. A blank OpenBus document named **System1.OpenBus** displays in the design window
2. Rename the new OpenBus document (with an **.OpenBus** extension) by selecting **File » Save As**. Navigate to the same folder as your project and type the name **SpinningVideo\_OB.OpenBus** and click on **Save**.

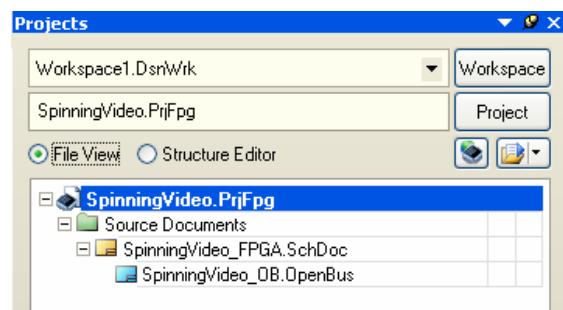


Figure 2. Projects Panel with FPGA project and newly created documents

## Defining an FPGA system using OpenBus

OpenBus is a new way of doing system-level FPGA design. It offers a much lighter interface than schematic based implementations but without becoming lightweight in its capabilities. By automatically taking care of much of the low-level detail, it lets you focus on the high-level system and interconnection of major components. You'll find all of the components you need in the **OpenBus Palette** panel. You can display the panel by clicking on the OpenBus panel control in the lower right portion of the main editor and then selecting the **OpenBus Palette** item from the popup menu.

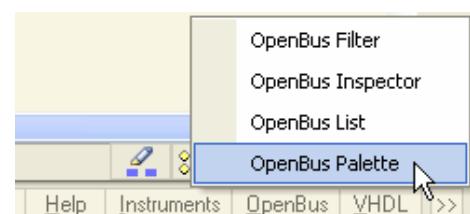


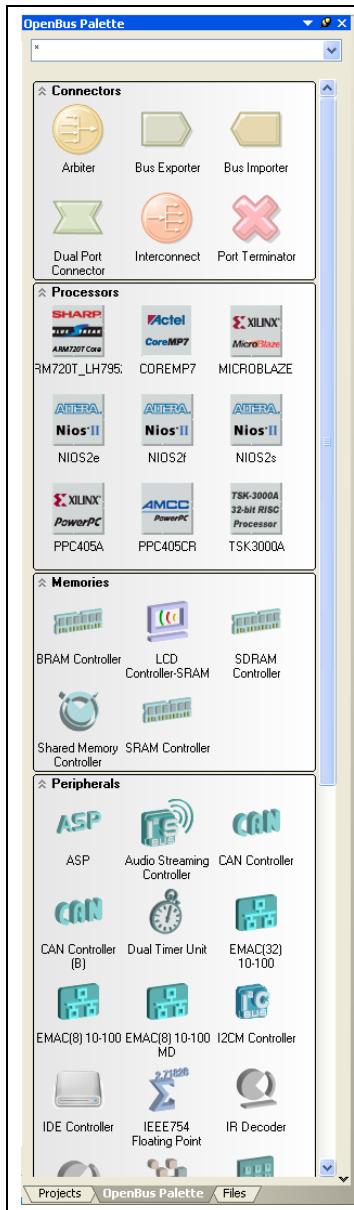
Figure 3. Accessing the OpenBus Panels

## Placing OpenBus Components

The **OpenBus Palette** panel contains all OpenBus components that can be used in an OpenBus document. They have been categorized in the palette into groups of Connectors, Processors, Memories and Peripherals. The subsections can be expanded or collapsed using the  or  icons respectively.

For this tutorial, we'll be using the following components:

Group	Item	Icon
Connectors	Arbiter	
Connectors	Interconnect	
Processors	TSK3000A	
Memories	SRAM Controller	
Peripherals	Port IO	
Peripherals	Video Capture Controller	
Peripherals	VGA 32-Bit TFT Controller	
Peripherals	I2CM Controller	



The OpenBus Palette panel is shown on the left side of the interface. It contains several sections: Connectors (Arbiter, Bus Exporter, Bus Importer, Dual Port Connector, Interconnect, Port Terminator), Processors (ARM720T Core, COREMP7, MICROBLAZE, Nios® II, PowerPC, PPC405A, PPC405CR, TSK3000A), Memories (BRAM Controller, LCD Controller-SRAM, SDRAM Controller, Shared Memory Controller), Peripherals (ASP, Audio Streaming Controller, CAN Controller, CAN Controller (B), Dual Timer Unit, EMAC(8) 10-100, IDE Controller, IEEE754 Floating Point, IR Decoder, I2CM Controller, I2S, MAC(32) 10-100, MD, Projects, OpenBus Palette, Files).

Figure 4. OpenBus components required for this tutorial

To place an OpenBus component onto an OpenBus Document:

1. Select the OpenBus Component that you want to place by left-clicking on its icon once in the **OpenBus Palette** panel.
  2. The component will be locked to the mouse cursor. At this point you can use the **Spacebar** to rotate the component or the **X** or **Y** keys to flip the component along the X or Y axis respectively.
  3. Move the mouse to where you want the component placed and left-click the mouse once again to place it. The cursor will remain in placement mode so that you can continue to place several more instances of the same components if required.
  4. Press the **Esc** key or right-click the mouse to exit the placement mode.
  5. Continue placing the components indicated in Figure 4 as per the OpenBus diagram of Figure 5.
- To edit the names of the OpenBus components that you have placed:
1. Click once on the text associated with the OpenBus component that you want to rename. This selects the text.
  2. Click a second time on the text or press the **F2** key to enter the edit text mode.
  3. Edit the text as desired.
  4. Press the **Enter** key or click on something else in the editor window to leave the editing mode and keep your changes.

## Connecting OpenBus Components

In order to control the flow of data between the components on your OpenBus document, you will need to place connection links between them. These links indicate bus connections between master ports  and slave ports . The arrow on the connection link indicates the direction of control.

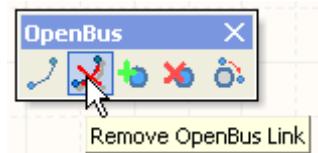
To place a connection link between a master and slave port:

1. Select **Tools » Link OpenBus Ports** or click on the Link OpenBus Devices icon in the **OpenBus** toolbar.
2. Click the master port that you want to create the link from
3. Click on the slave port that you want to be controlled.
4. Repeat steps 2 & 3 for any additional links that you wish to create. To exit link placement mode, hit the **Esc** key or right-click the mouse.



To remove a connection link between a master and slave port:

1. Select **Tools » Remove OpenBus Link** or click on the Remove OpenBus Link icon in the **OpenBus** toolbar.
2. Hover the mouse over the link that you wish to remove until it changes color. Click the left-mouse button to execute the link removal.
3. Repeat step 2 for any additional links that you wish to remove. To exit the link removal mode, hit the **Esc** key or right-click the mouse.



## Interconnect and Arbiter Components

Because connection links can only be made between a single master and a single slave, OpenBus Interconnect and Arbiter components are required to allow you to connect multiple components together.



OpenBus Interconnect components have a single slave port and one or more master ports. This allows a master device (connected to the OpenBus Interconnect's slave port) to control multiple other slave devices (connected to the OpenBus Interconnect's master ports).



OpenBus Arbiters are the complement to Interconnect Components. They allow multiple master components to control a single device such as when you need to connect multiple master components to a single block of memory. The Arbiter component is responsible for coordinating accesses between the competing masters. You can control how it performs its arbitration by right-clicking the component and accessing its configuration options.

To add an additional port to an OpenBus Interconnect or Arbiter component:

1. Select **Tools » Add OpenBus Port** or click on the Add OpenBus Port icon in the **OpenBus** toolbar.
2. Hover the mouse over an existing port on the component you wish to add a new port to. A red line will appear indicating where the port will be added.
3. Click to add the new port.
4. Repeat steps 2 & 3 for any additional ports that you wish to create. To exit the port placement mode hit the **Esc** key or right-click the mouse.



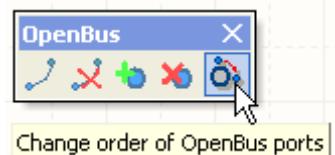
To remove a port from an OpenBus Interconnect or Arbiter component:

1. Select **Tools » Remove OpenBus Port** or click on the Remove OpenBus bus port icon in the **OpenBus** toolbar.
2. Hover the mouse over the port that you wish to remove until a red cross appears. Click the left-mouse button to execute the port removal.
3. Repeat step 2 for any additional ports that you wish to remove. To exit the port removal mode, hit the **Esc** key or right-click the mouse.



To change the position of ports on an OpenBus Interconnect or Arbiter component:

1. Select **Tools » Reorder OpenBus Ports** or click on the Reorder OpenBus ports icon in the **OpenBus** toolbar.
2. Hover the mouse over the port that you wish to move until a red circle is drawn around it. Click the left-mouse button to begin the move.
3. Move the mouse cursor over another port on the component. A red line will appear at the location where the port will be moved to. Left click the mouse to execute the move.
4. Repeat steps 2 & 3 for any additional ports that you wish to move. To exit the port reorder mode, hit the **Esc** key or right-click the mouse.



## Completing the OpenBus System

1. Complete the creation of the OpenBus system using the techniques outlined above. The completed OpenBus system can be seen in Figure 5.
2. Save your work.

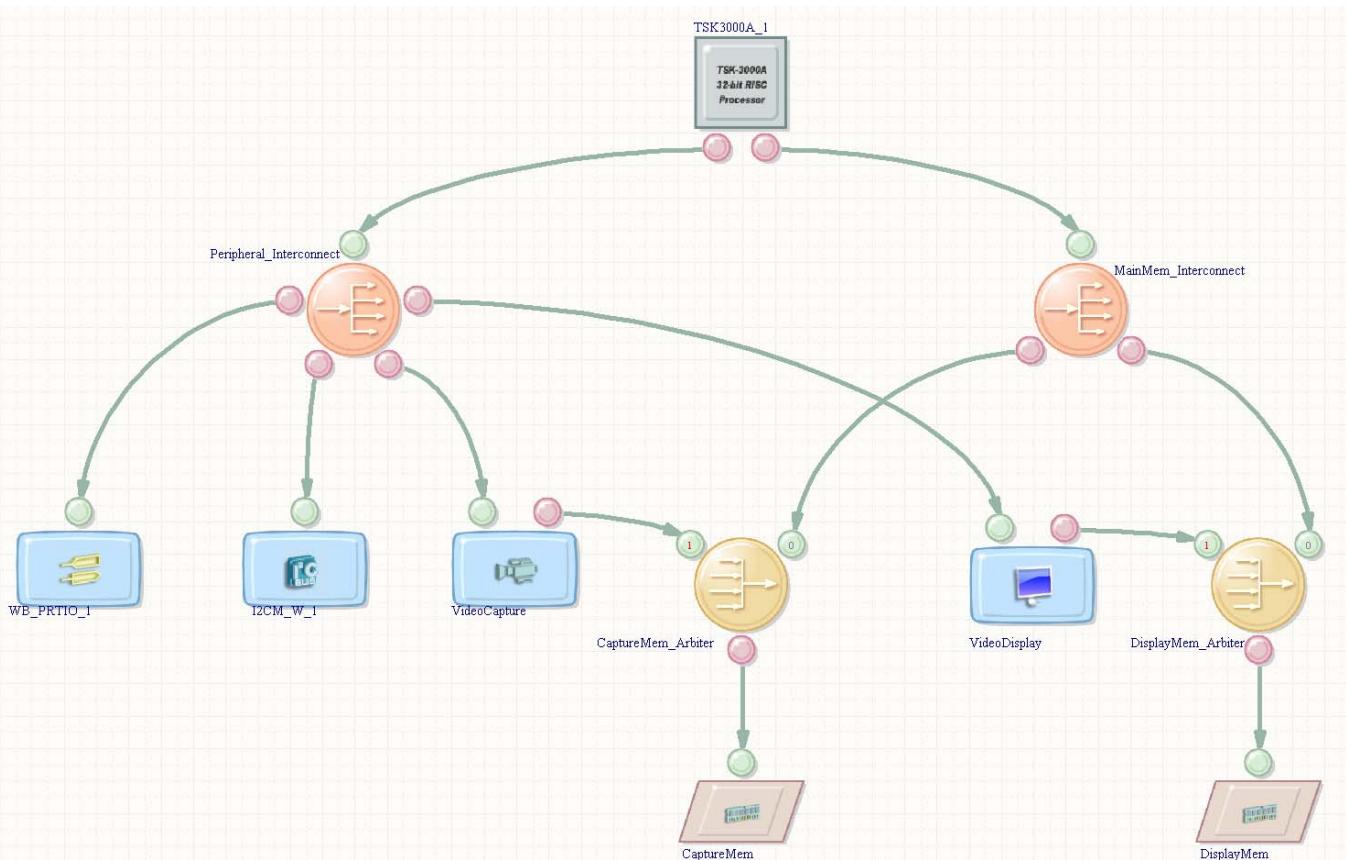


Figure 5. Completed OpenBus design for this tutorial

## Configuring OpenBus Components

The vast majority of components found in the OpenBus Palette translate directly to similarly named components found within the FPGA Peripherals and FPGA Processor libraries used for schematic-based FPGA design. In the same way that several of the components in the schematic-based libraries are configurable, so too are their OpenBus counterparts. In this tutorial, we are using several configurable components that will need to be adjusted.

### Configure Peripheral Components

To configure the Port I/O component for this tutorial:

1. Right-click the WB\_PRTIO\_1 component and select the **Configure WB\_PORTIO\_1 (Port IO) ...** item.
2. In the *Configure OpenBus Port I/O* dialog box:
  - Set the port **Kind** to **Output**.
  - Set the **Port Count** to **1**.
  - Set the **Bus Width** to **8**.
3. Click **OK** to save the changes.

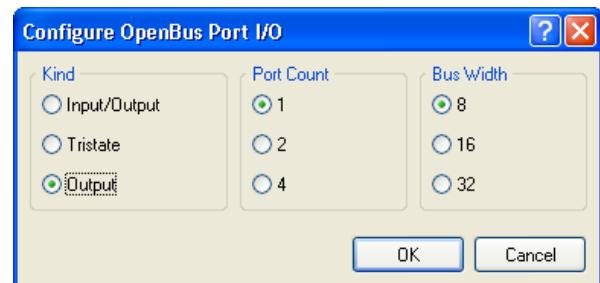


Figure 6. Configuring the IO Port peripheral

### Configure Memory Controllers

To Configure the Video Capture Memory component:

1. Right-click the CaptureMem component and select the **Configure CaptureMem (SRAM Controller) ...** item.
2. In the *Configure (Memory Controller)* dialog box:
  - Set the **Memory Type** to **Asynchronous SRAM**.
  - Set the **Size of Static RAM array** to **1 MB (256K x 32-bit)**.
  - Set the **Memory Layout** to **1 x 32-bit Wide Device**.
3. Click **OK** to save your changes.

To configure the Video Display Memory component:

1. Right-click the DisplayMem component and select the **Configure DisplayMem (SRAM Controller) ...** item.
2. In the *Configure (Memory Controller)* dialog box:
  - Set the **Memory Type** to **Asynchronous SRAM**.
  - Set the **Size of Static RAM array** to **1 MB (256K x 32-bit)**.
  - Set the **Memory Layout** to **2 x 16-bit Wide Devices**.
3. Click **OK** to save your changes.

### Configure Memory Arbiters

To configure the Capture Memory Arbiter component:

1. Right-click the CaptureMem\_Arbiter component and select the **Configure CaptureMem\_Arbiter (Arbiter) ...** item.
2. In the *Configure OpenBus Arbiter* dialog box:
  - Set the **Type** to **Priority**.
  - Set the **Master With No Delay** to be same slave port as the VideoCapture component is connected to (**S1**).
3. Click **OK** to save your changes.



Figure 7. Configuring an Arbiter component

To configure the Display Memory Arbiter component:

1. Right-click the DisplayMem\_Arbiter component and select the **Configure DisplayMem\_Arbiter (Arbiter) ...** item.
2. In the *Configure OpenBus Arbiter* dialog box:
  - Set the **Type** to **Priority**.
  - Set the **Master With No Delay** to be same slave port as the VideoDisplay component is connected to (**S1**).
3. Click **OK** to save your changes.

## Configure the Processor

To configure the TSK3000 processor component:

1. Right-click the TSK3000 component and select **Configure TSK3000A\_1 (TSK3000) ...**
2. In the *Configure (32-bit Processors)* dialog box:
  - Confirm that the processor listed in the drop down box in the upper right is **TSK3000**.
  - Set the **Internal Processor Memory** to **8 K Bytes (2K x 32-Bit Words)**.
  - Set the **Multiply/Divide Unit (MDU)** to **Hardware MDU**.
  - Set the **On-Chip Debug System** to **Include JTAG-Based On-Chip Debug System** and **Disable Breakpoints on Hard Reset**.
3. Click **OK** to save your settings and close the dialog box.

## Managing the Memory Map

One of the key benefits of developing your design with OpenBus is the level of automation that OpenBus brings to the management of the system memory map. Ultimately all peripherals and memory devices sit within a 32-bit memory space that spans 4GBytes. To make the management of this memory space easier, OpenBus intelligently interprets the design and automatically allocates memory spaces for each of the peripherals and memory controllers. In most situations these memory allocations will be sufficient but in some rare cases you may wish to manually edit the memory allocations yourself. You can still do this with OpenBus.

### Configuring Memory using Interconnect Components

Under normal circumstances, the interconnect component will probe the settings and memory requirements of each of its connected devices and will update the memory map automatically. You can see (and edit) the memory mapping of the interconnect component from the *Configure OpenBus Interconnect* dialog which can be accessed from the component's right-click menu.

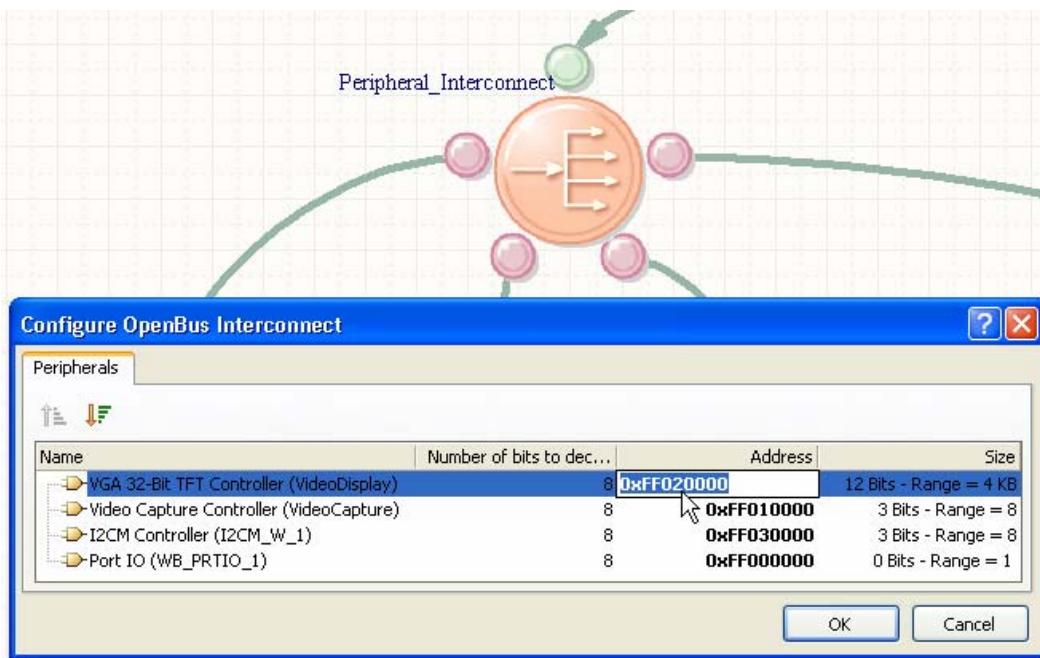


Figure 8. Viewing (and editing) the interconnect memory map

The Interconnect configuration information is automatically propagated to the memory map of the processor.

## Configuring Memory from within the Processor

In addition to the memory that is managed by each of the interconnect components within the OpenBus document, you can also centrally manage memory from the Processor's memory and peripheral configuration dialog boxes. These can be accessed by right-clicking the processor.

The **Configure Processor Memory** dialog box provides a pictorial representation of where the peripherals and memory controllers will be positioned within the processor's memory. You can manually control the memory map using the grid control at the bottom of the dialog but this is usually unnecessary. A far simpler option is to check the **Automatically import when compiling** checkbox. This will ensure that all OpenBus memory settings are incorporated and synchronized in the design each time it is compiled.

Checking the **hardware.h (C Header File)** option will cause a hardware.h header file to be created when the project is compiled. This header file will be automatically added to any linked embedded projects and will provide macros that define where each of the peripherals and memory devices sit within memory. By ensuring these two checkboxes are checked, any changes you make to the OpenBus document will be propagated through to the embedded project as well.

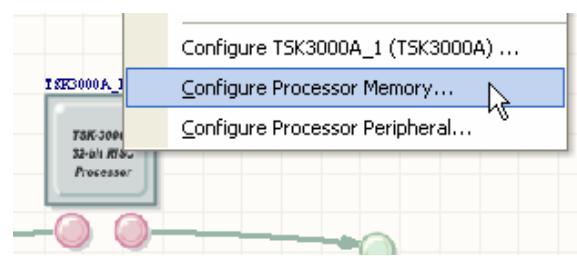


Figure 9. Configuring the processor memory

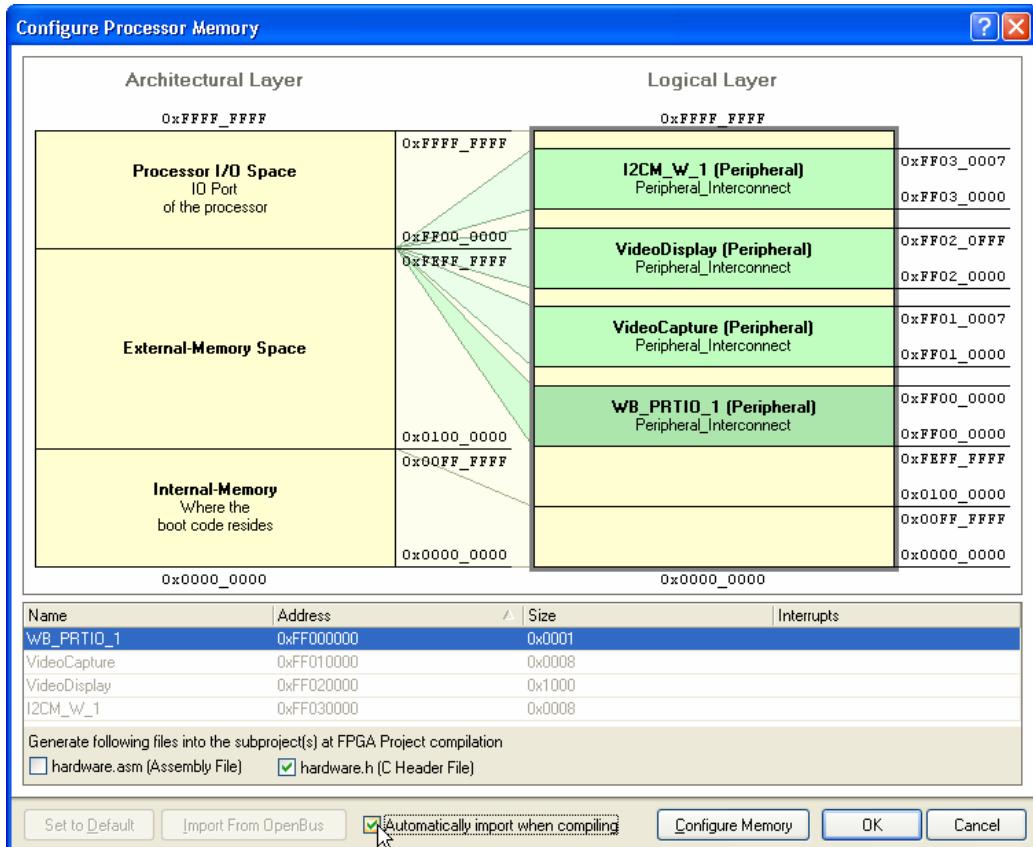


Figure 10. Controlling how memory configuration information is propagated throughout the design.

To set the project to automatically import the settings from the OpenBus document:

1. Right-click the TSK3000 processor and select **Configure Processor Memory ...**
2. In the *Configure Processor Memory* dialog box, check the **hardware.h (C Header File)** checkbox.
3. Check the **Automatically import when compiling** checkbox also.
4. Click the **Configure Peripherals** button to change views. If asked if you want to save the configuration before moving to configuring the peripherals, click **Yes**.
5. Check the **Automatically import when compiling** checkbox to import the peripheral memory map when the project is compiled.
6. Click **OK** to exit.

## Linking the OpenBus Document to its Parent Schematic

As mentioned right back at the beginning of this tutorial, the top level document in an FPGA project needs to be a schematic. Now that we have created an OpenBus document, we must now link that document back to the top level schematic sheet that we created earlier on.

Before linking the documents, it's worth taking just a moment to explain how the underlying signals in the OpenBus document get exposed to the top level schematic. All of the signals used for the bus interconnects (links) will generally remain hidden within the OpenBus document but there are many other signals that are not immediately visible on the OpenBus document that will need to be exposed. To see a list of these signals you'll need to use the **OpenBus Signal Manager**.

### Using the OpenBus Signal Manager

The **OpenBus Signal Manager** can be accessed from the **Tools** menu and it allows you to take finer control over which signals are to be exposed externally to the OpenBus document. A picture of this dialog is given in Figure 11.

The **Clocks** and **Resets** tabs will rarely need your attention as the default settings for these are usually adequate. The **Interrupts** tab will need your attention if you are planning on using any peripherals as interrupt sources. From this dialog you can allocate interrupts to the available interrupt channels on the main processor. In this design, interrupts won't be needed and so we can leave things in the default (unconnected) state.

The list of signals in the **External connection summary** can not be edited directly however this dialog serves as an excellent reference. All of the signals listed in this dialog will be exported to the parent schematic. The signals are grouped according to the component that controls them which makes it much easier to identify and locate the source of the different signals. For instance, when you link an OpenBus document to a parent schematic, it may not be immediately apparent where certain signals on the sheet symbol have come from. It is in this dialog box that you will find your answers.

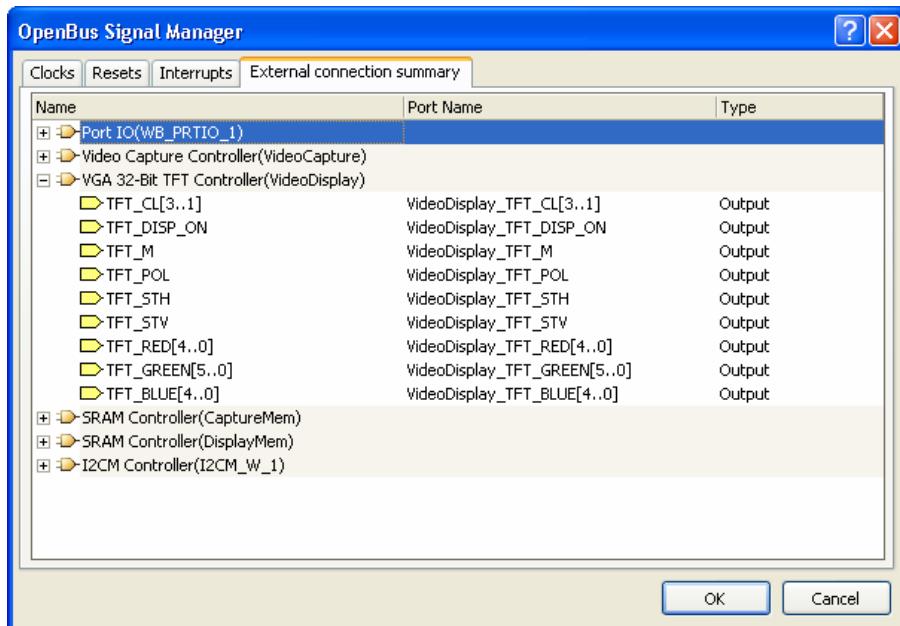


Figure 11. Using the OpenBus Signal Manager to identify the source of OpenBus signals.

### Creating a sheet symbol from the OpenBus Document

To link the OpenBus document to a parent schematic, you need to create a sheet symbol from the OpenBus document and place it on the parent schematic.

1. Open SpinningVideo\_FPGA.SchDoc
2. Select **Design >> Create sheet symbol from sheet or HDL**.
3. When the *Choose Document to Place* dialog box appears, select the SpinningVideo\_OB.OpenBus document and click **OK**.
4. A large sheet symbol will be attached to the cursor. Position it where you want to place it on the schematic page and click once to commit the placement.

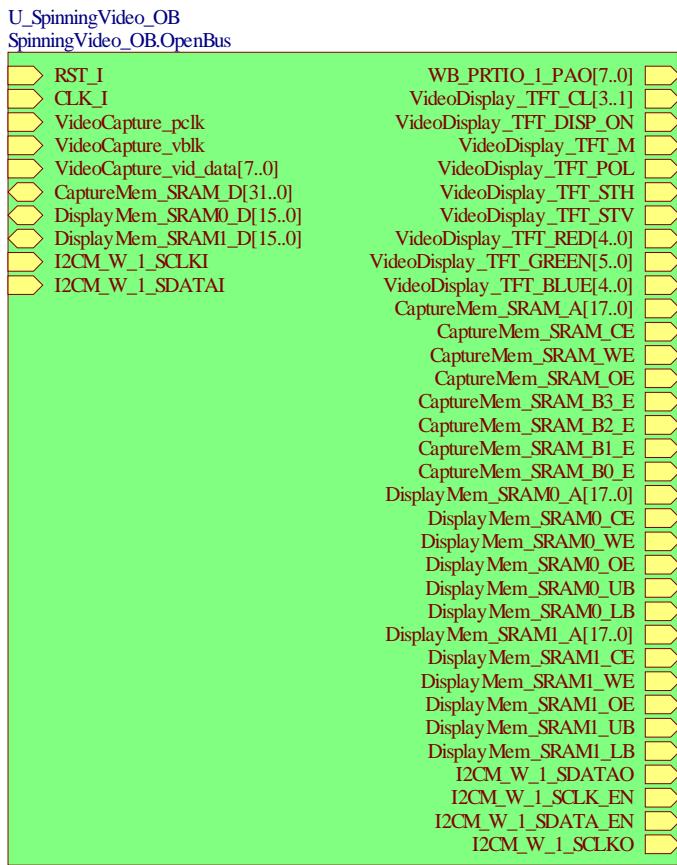


Figure 12. Creating a sheet symbol from an OpenBus document

The sheet entries on the newly placed sheet symbol have been loosely grouped with inputs on the left and outputs on the right. You must now go through a process of unraveling all of these sheet entries so that you can connect them to the port plugins on the NanoBoard more easily.

## Wiring up the Top Level Schematic

To line the sheet entries up with the port plugins and to complete the top level schematic, you'll need to place a number of components. You'll find these libraries in the `Libraries\FPGA` folder of your Altium Designer installation.

1. From the `FPGA_NB2DSK01_Port-Plugin.IntLib`:
  - `CLOCK_BOARD`
  - `LED`
  - `NEXUS_JTAG_CONNECTOR`
  - `SHARED_MEM_DAUGHTER`
  - `SRAM_DAUGHTER0`
  - `SRAM_DAUGHTER1`
  - `TEST_BUTTON`
  - `TFT_LCD`
2. From the `FPGA_Peripheral Board 01_Port-Plugin.IntLib`:
  - `VIDEO_INPUT`
  - `VIDEO_INPUT_CTRL`
3. From the `FPGA_Generic.IntLib`:
  - `INV`
  - `IOBUF (x2)`
  - `J4S_4B`
  - `NEXUS_JTAG_PORT`

4. A completed version of the SpinningVideo\_FPGA.SchDoc schematic is given in Figure 13. Use this as a guide to position the port plugin components around the sheet symbol and then reorder the sheet entries so that they will line up nicely with them.
5. Once you have completed wiring up the schematic, select **Tools » Annotate Schematics Quietly...** to annotate the design by giving each component a unique designator.
6. Compile the design by selecting **Project » Compile FPGA Project SpinningVideo.PrjFpg**. Fix any compilation or wiring errors as necessary and save your work.

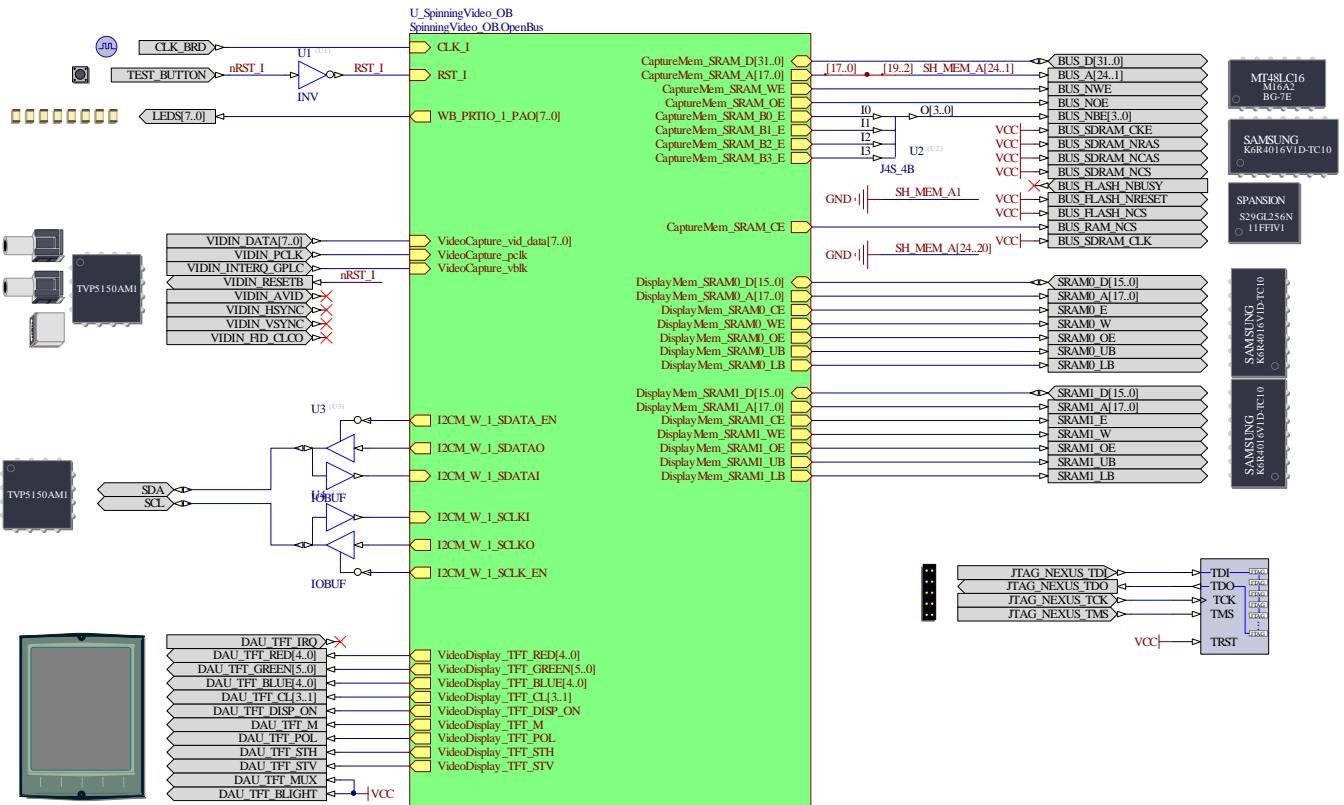


Figure 13. The Completed top-level schematic Constraining the Design

At this point we have completed the bulk of the FPGA design but there is one additional step that we need to go through before we can run it on the Desktop NanoBoard. Constraining an FPGA design is the process of defining the specific FPGA pins that you want each of the signals in your design to appear on. This is an important step as it will ensure that the FPGA design is able to interact with NanoBoard resources that have been hardwired to the FPGA daughter board.

When defining constraints, it is possible to hardcode them into the top-level schematic sheet but this is not advisable. The reason for this is because it binds the design to a specific device and limits your ability to retarget a different FPGA should the need arise. A much better approach is to store constraint information in a separate location to the schematic. Altium Designer implements this approach using a set of pre-built and user-definable constraint files which can be added to the FPGA project.

## Auto-configuring projects running on the Desktop NanoBoard

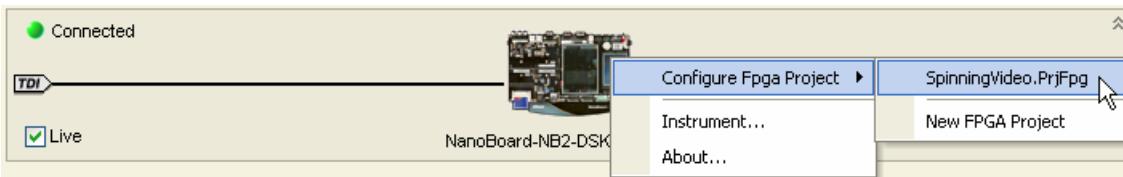


Figure 14. Creating a configuration automatically

In order to make the process of targeting your design to the Desktop NanoBoard, Altium Designer includes a handy autoconfiguration feature. By utilizing some smarts that have been built into the NanoBoard's firmware, Altium Designer is able to probe the Desktop NanoBoard and determine exactly what daughter and peripheral boards are connected. A set of pre-defined constraint files will then be loaded and grouped together into a configuration that targets your specific hardware setup.

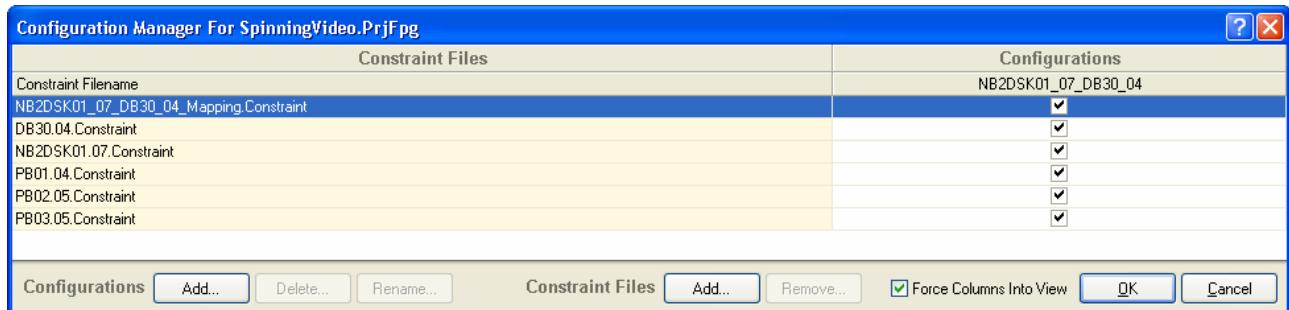


Figure 15. Example of a configuration created from the Autoconfigure utility

To auto-configure your FPGA design to run on a Desktop NanoBoard:

1. Make sure your Desktop NanoBoard is connected to your PC and powered on.
2. Select **View** » **Devices View** or click on the **Devices View** icon  in the toolbar.
3. In the **Devices View**, ensure that the **Live** checkbox is checked. You should see a picture of the Desktop NanoBoard in the upper region of the display.
4. Right-click the Desktop NanoBoard icon and select **Configure FPGA Project** » **SpinningVideo.PrjFpg**.
5. Altium Designer will take just a few moments to probe the Desktop NanoBoard and create a new configuration. Click **OK** to accept the new configuration.

You may notice that a new **Settings** folder has been added to the project. In this folder you will find a **Constraint Files** folder with all of the newly added constraint files.

Several of the files will have a 'shortcut'  symbol. Altium Designer uses this notation to indicate files which are not stored in the main project folder. These particular files are all pre-defined constraint files that are shipped with Altium Designer. They are specific to the various peripheral and daughter boards that they represent and should NOT be edited as changes made to these files will affect all other projects that you build for the Desktop NanoBoard.

The constraint file that has been highlighted in Figure 16 was automatically created by the auto-configure process and is stored with the project. This file defines where the peripheral boards are located on the Desktop NanoBoard.

The auto-configuration process deals with the mapping of ports defined on the top-level FPGA schematic and their target FPGA pins. There are, however, additional constraints (such as the clock frequency) that are important for the design but which can not be handled automatically. In order to capture this information, it is best to create another constraint file that is reserved for this information and add it to the configuration.

To create a new constraint file and add it to the configuration:

1. Right-click the SpinningVideo.PrjFpg project in the **Projects** panel and select **Add New to Project** » **Constraint File**.
2. Select **File** » **Save As ...** to save the file. Give it a meaningful name such as **MyConstraints.Constraint** and click **OK**.
3. Right-click the SpinningVideo.PrjFpg project in the **Projects** panel and select **Configuration Manager ...**.
4. Locate **MyConstraints.Constraint** in the **Constraint Files** column and check the box in the **Configurations** column to add it to the existing configuration.
5. Click **OK** to close the **Configuration Manager** and save your changes.

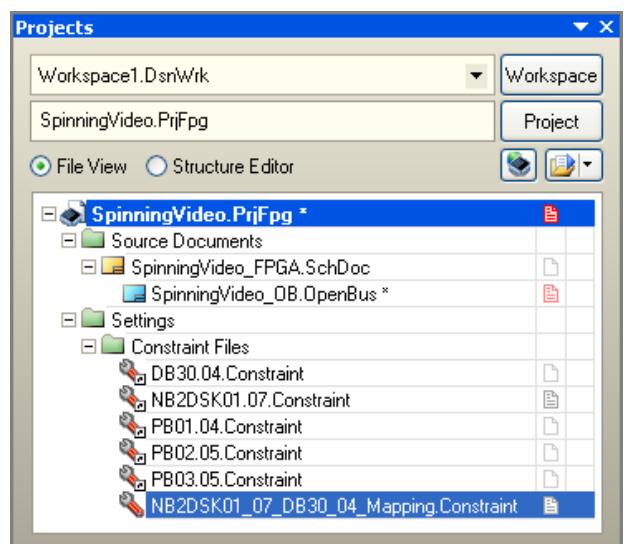


Figure 16. Constraint files that have been added to the project by the autoconfigure utility.

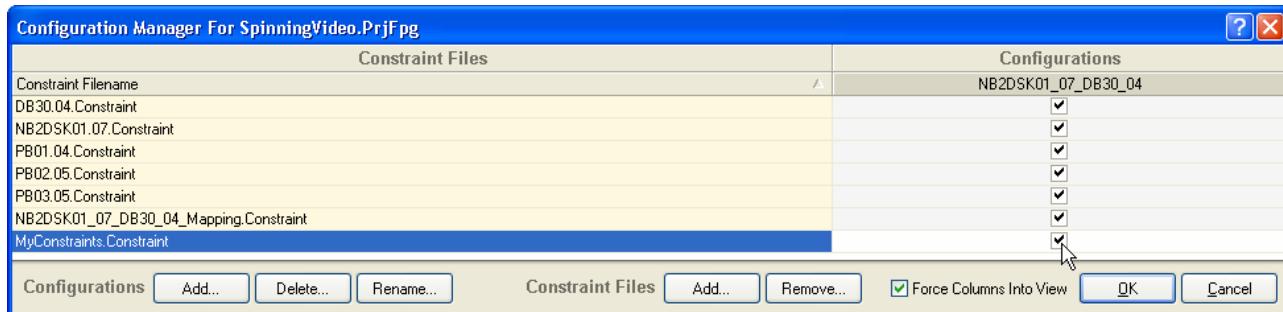


Figure 17. Adding your own constraint file to the configuration

To add a clock constraint to the CLK\_BRD signal:

1. Open `MyConstraints.Constraint`.
2. Select **Design » Add/Modify Constraint ... » Port ...**.
3. In the *Add/Modify Port Constraint* dialog:
  - Set the **Target** to CLK\_BRD
  - Set the **Constraint Kind** to FPGA\_CLOCK\_FREQUENCY
  - Set the **Constraint Value** to 50MHz.
4. Click **OK** to close the *Add/Modify Port Constraint* dialog.
5. Observe that a new constraint record has been added to `MyConstraints.Constraint`. Save your work.

```
Record=Constraint | TargetKind=Port | TargetId=CLK_BRD | FPGA_CLOCK_FREQUENCY=50MHz
```

Figure 18. A clock frequency constraint

## Building the FPGA design

Once the FPGA design has been defined along with its constraints, you are now ready to build it. Building an FPGA design is the process of compiling and synthesizing your entire FPGA design into a configuration bit file that can be downloaded and run from the target FPGA device. Altium Designer standardizes the way you build an FPGA design so that it is vendor independent. You'll recall that a copy of the Vendor tools for the specific device you are targeting was listed in the "What you'll need" section of this tutorial. Altium Designer needs these vendor tools in order to place and route the design but the interaction with these back end tools will be largely transparent to the user.

To build an FPGA design:

1. Make sure your Desktop NanoBoard is connected to your PC and powered up.
2. Select **View » Devices View** or click on the **Devices View** icon  in the toolbar.
3. Ensure that the **Live** checkbox is checked. You should see a picture of the Desktop NanoBoard in the upper region of the display and an icon of the Spartan3 FPGA in the middle region.
4. In the drop down list just below the Spartan3 icon, ensure that the **SpinningVideo / NBD2DSK01\_07\_DB30\_04** project / configuration pair is selected.
5. Locate the **Compile, Synthesize, Build, Program FPGA** buttons running left to right just below the Desktop NanoBoard icon. As this is the first time you have built your design, the colored indicators on each of the buttons will appear RED. Click once on the words **Program FPGA** to begin the build process.
6. As the build process progresses, the colored indicator from each stage will turn yellow while it is processing and then green when completed successfully. The process of building the design may take several minutes to complete. You can observe the progress of the build from the **Messages** and **Output** panels which can be accessed from the **System** panel tab in the lower right section of the Altium Designer workspace.

## Capturing Video the Easy Way – Programming FPGAs for flexible, quick applications development

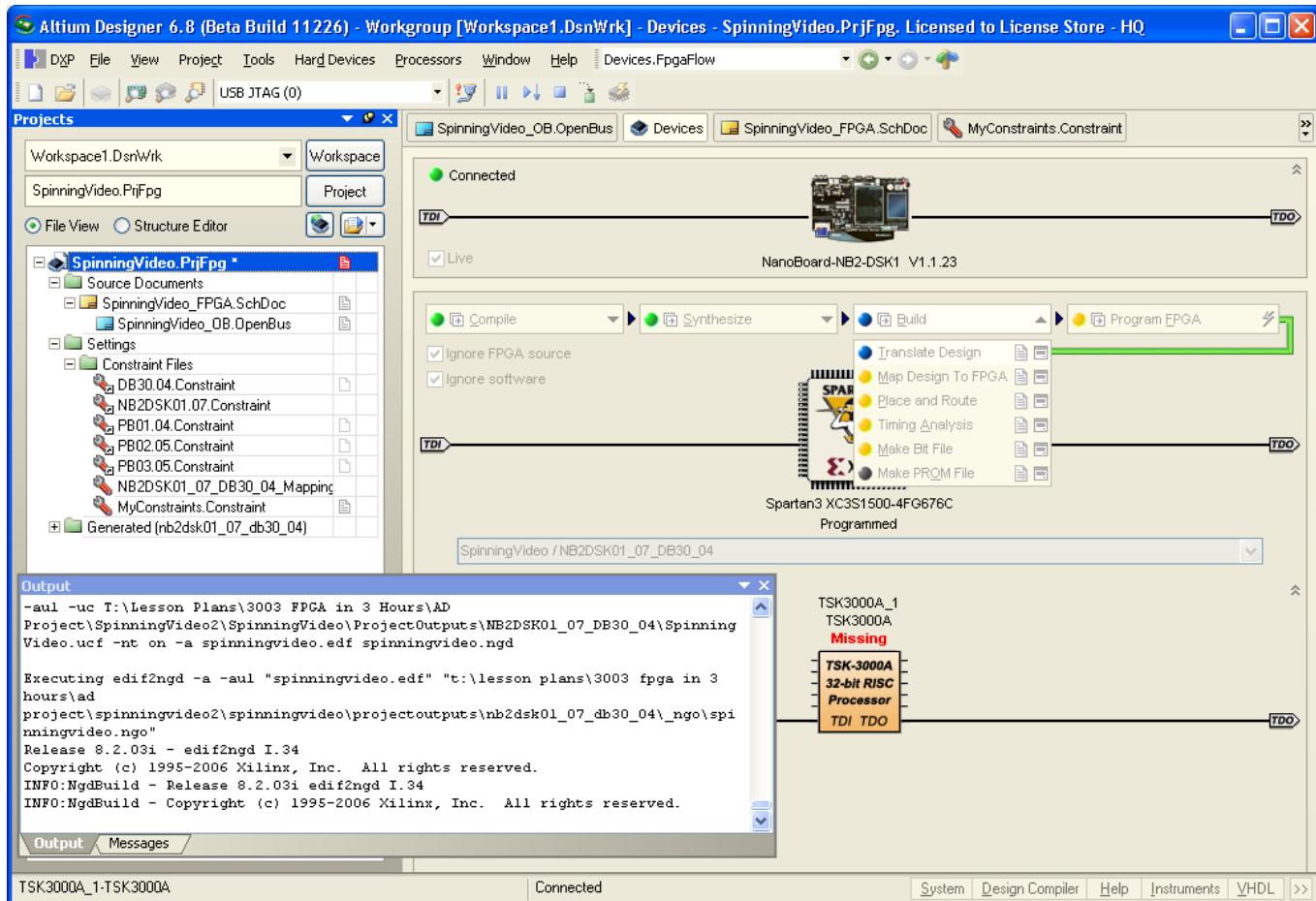
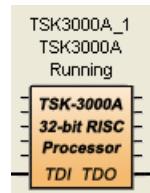


Figure 19. Running the FPGA build flow

1. If any errors occur you will need to rectify them before you can proceed. Try to locate the source of the error by retracing your steps through the instructions of the tutorial.
2. A summary dialog will be displayed once the design has been built and downloaded successfully. Click **OK** to close this dialog.

Once the FPGA design has been downloaded to the Desktop NanoBoard, you should notice that the status of the TSK3000 processor has changed from **Missing** to **Running**.

We can now begin work on the embedded code that this processor will run.



## Developing the Embedded Code

Altium Designer uses an Embedded Project as the container for all of the code that is to execute on a given target.

To create a new Embedded Project:

1. Select **File** » **New** » **Project** » **Embedded Project** from the menus, or click on **Blank Project (Embedded)** in the **New** section of the **Files** panel.
2. The **Projects** panel will display a new Embedded project with the default name `Embedded_Project1.PrjEmb`. Select **File** » **Save Project** or right-click the project in the **Projects** panel and select the **Save Project** item. Save the file as `VideoSpin.PrjEmb`. If you want to keep your Embedded project documents separate from your FPGA project documents, you may wish to save the Embedded Project in a subfolder called `Embedded` under your FPGA project folder.

## Adding the source code to the Embedded Project

When a new embedded project is first created, it will be created as an empty container. You must then add or create the relevant source files to the project. Altium Designer can compile C source files, C header files, or Assembly files as part of your project.

To create a new C file and add it to the project:

1. Select **File » New » C Source Document**, or right-click the embedded project in the **Projects** panel and select **Add New to Project » C File**. A blank text document named **Source1.c** will be added to the Embedded Project and displayed in the main editor window.
2. Rename the newly created file (with a **.c** extension) by selecting **File » Save As**. Navigate to the same folder as your embedded project and type the name **Main.C** and click on **Save**.

## Linking an Embedded Project to its Target Processor

An Embedded Project can be developed in isolation but pretty soon you're going to want to run it on a target processor. Altium Designer gives you the ability to link your embedded project to an FPGA project containing an embedded processor.

To link an Embedded Project to its target processor:

1. Make sure both the Embedded Project and the FPGA Project that contains the target processor are both loaded in the **Projects** panel.
2. Select the **Structure Editor** mode in the **Projects** panel.
3. Left-click and drag the Embedded Project over the top of the FPGA project. Any valid processor targets will be highlighted in light blue. Drop the project on the **TSK3000A\_1** processor.
4. Switch the **Projects** panel back to **File View**. Observe that the hierarchy of the projects have been updated placing **VideoSpin.PrjEmb** as a child of **SpinningVideo.PrjFpg**.

After you've linked projects, you may also notice that a new C header file has been added to the Embedded Project. This file, **hardware.h**, was created by the FPGA project when it was compiled and, according to the options that were set in Figure 10, has been added to the linked embedded project.

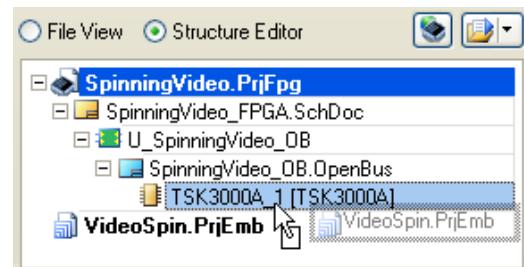


Figure 20. Linking an Embedded Project to a processor

## Writing some C Source Code

Now that our Embedded Project has been linked to a hardware platform that it can execute from, we are ready to start writing some C code. We'll take things slowly to begin with and build things up a little later.

To add some simple code to the Embedded Project:

1. Open the **hardware.h** file that is now part of the **VideoSpin.PrjEmb** project. Observe that an entry for the Port IO component's base address has been made.

```
#define Base_WB_PRTIO_1          0xFF000000
#define Size_WB_PRTIO_1           0x00000001
```

2. Open the **Main.C** file and enter the following source code:

```
#include "hardware.h"

#define LEDS    (*(unsigned char*)Base_WB_PRTIO_1)

void main(void)
{
    LEDS = 0x55;
}
```

3. Compile and download the source code to the TSK3000 processor running on the Desktop NanoBoard by pressing the icon in the main toolbar. After the program has been downloaded, you should see the LEDs on the NanoBoard light up with the value **55h**.



## Developing the Complete Application

At the beginning of this tutorial we indicated that we would create a complete application that is capable of reading images from a composite video source and displaying them on the TFT screen of the Desktop NanoBoard. So far we have laid all of the foundations for this application and all that remains is the final source code. Before we can embark on this, there are a couple of minor tweaks we'll do to the embedded project to define different memory sections for the video capture and display memory respectively.

To create a memory section in the embedded project:

1. With a file from the Embedded Project focused in the main editor, select **Project » Project Options** or right-click the **VideoSpin.PjEmb** project in the **Projects** panel and select **Project Options**.
2. Select the **Configure Memory** tab and observe that entries for **CaptureMem** and **DisplayMem** exist.
3. Select the **Sections/Reserved Areas** tab of the *Options for Embedded Project VideoSpin.PjEmb* dialog.
4. Click the **Add Section** button.
5. In the *Section* dialog box:
  - Set the **Name** to **.bss.capture**
  - Set the **Location** to **mem:CaptureMem**
6. Click **OK** to close the *Section* dialog box and observe that the new memory section has been added.
7. Click the **Add Section** button again
8. In the *Section* dialog box:
  - Set the **Name** to **.bss.display**
  - Set the **Location** to **mem:DisplayMem**
9. Click **OK** to close the *Section* dialog box and observe that the new memory section has been added.
10. Click **OK** to close the *Options for Embedded Project VideoSpin.PjEmb* dialog.
11. With the different memory sections labeled, we can now use this label as a qualifier when we create an array for the video capture and video display buffers. Open the **Main.C** file and add the following lines of code:

```
// Display memory
#pragma section .bss=.display
volatile pixel_t display[TFT_XRES * TFT_YRES];
#pragma endsection

// Capture memory
#pragma section .bss=.capture
volatile pixel_t capture[VIDEO_BUFFER_SIZE];
#pragma endsection
```

## Interacting with the Peripherals

As well as providing a number of high-level peripheral devices, Altium Designer also includes software routines that make interfacing with those peripherals much simpler. By using the Device Software Framework (DSF), you can skip over having to deal with low level interfaces and use higher-level control routines.

The DSF system is still under development and some of the documentation is still being produced. But all of the source code can be found in the **System\Tasking\dsf** folder under your Altium Designer installation. Figure 21 contains the complete listing of the **Main.C** file. It is based on the DSF implementation as of Altium Designer release 6.8.

```
#define DSF_IMPLEMENT
#include <dsf_system.h>

bt656_context_t capture_settings;
vga_context_t display_settings;

// Resolution of raw video input, for PAL-B/G
#define PALWIDTH 720
#define PALHEIGHT 580 // 625
#define VIDEOWIDTH 450
#define VIDEOHEIGHT 450
#define VIDEO_BUFFER_SIZE (VIDEOWIDTH * VIDEOHEIGHT * sizeof( pixel_t ) )
```

```

// Video: defines for centering
#define VIDEO_X0      (VIDEOWIDTH / 2)
#define VIDEO_XMIN    (VIDEO_X0 - VIDEOWIDTH)
#define VIDEO_XMAX    (VIDEOWIDTH - VIDEO_X0)

#define VIDEO_Y0      (VIDEOHEIGHT / 2)
#define VIDEO_YMIN    (VIDEO_Y0 - VIDEOHEIGHT)
#define VIDEO_YMAX    (VIDEOHEIGHT - VIDEO_Y0)

// TFT type
typedef uint16_t pixel_t;

// Resolution of TFT
#define TFT_XRES     240
#define TFT_YRES     320

// TFT: defines for centering
#define TFT_X0      (TFT_XRES / 2)
#define TFT_XMIN    (TFT_X0 - TFT_XRES)
#define TFT_XMAX    (TFT_XRES - TFT_X0)

#define TFT_Y0      (TFT_YRES / 2)
#define TFT_YMIN    (TFT_Y0 - TFT_YRES)
#define TFT_YMAX    (TFT_YRES - TFT_Y0)

bt656_context_t capture_settings;
vga_context_t display_settings;

// Display memory
#pragma section .bss=.display
volatile pixel_t display[TFT_XRES * TFT_YRES];
#pragma endsection
// Capture memory
#pragma section .bss=.capture
volatile pixel_t capture[VIDEO_BUFFER_SIZE];
#pragma endsection

#define PI   3.141592654

#define LEDS          (*(unsigned char*)Base_WB_PRTIO_1)

void main(void)
{
    uint32_t line;
    uint32_t pixel;
    pixel_t* inpixel;
    pixel_t* outpixel;
    uint8_t count = 0;

    //Main Processor clock rate
    __clocks_per_sec = 50000000; // Assuming we run at 50 MHz

    //Initialize TFT display
    vga16_init(& display_settings, Base_VideoDisplay, (uintptr_t) display, CLOCKS_PER_SEC,
    VGA16_TFT);

    /* I2C */
    i2c_init(Base_I2CM_W_1);

    // Initialize video capture
    bt656_init(& capture_settings, Base_VideoCapture, Base_I2CM_W_1, (void *) capture,
    VIDEO_BUFFER_SIZE, BT656_RGB16, BT656_COMP1);
    bt656_set_linesize(& capture_settings, VIDEOWIDTH);
    bt656_set_scale(& capture_settings, 1, 1);
    bt656_set_framerate(& capture_settings, 1);
}

```

```
bt656_set_crop(& capture_settings, (PALWIDTH - VIDEOWIDTH) / 2, (PALHEIGHT - VIDEOHEIGHT) /  
2, (PALWIDTH - VIDEOWIDTH) / 2, (PALHEIGHT - VIDEOHEIGHT) / 2);  
bt656_set_buffer(& capture_settings, (uint32_t *) capture, VIDEO_BUFFER_SIZE);  
bt656_set_mode(& capture_settings, BT656_RUN);  
  
while (1)  
{  
    outpixel = display;  
    for (line = 0; line < TFT_YRES; line++)  
    {  
        for (pixel = 0; pixel < TFT_XRES; pixel++)  
        {  
            *outpixel = capture[(pixel + (VIDEOWIDTH-TFT_XRES)/2) + ((line + (VIDEOHEIGHT-  
TFT_YRES)/2) * VIDEOWIDTH)];  
            outpixel++;  
        }  
    }  
    LEDS = count++;  
}
```

Figure 21. Completed Video Capture and Display source code

## Considering your Deployment Options

Now that we've arrived at a functional design, let's examine how you might deploy your product in the field. Altium provides a range of deployment NanoBoards that you can either use entirely as an off-the-shelf solution or that you can customize with your own peripheral boards. Alternatively you can go for a fully custom PCB solution that makes selective use of existing NanoBoard circuit blocks and combines them together into a single design. The choice of deployment options will be influenced by a range of factors including cost, time to market, logistics, and form and fit constraints. While we can't tell you which solution will be best for your specific situation, we can present the range of options so that you can get a fair indication of their pros and cons.

### Level 1: Development of pure 'Device Intelligence'

Almost all designs will begin at this level. The focus of development is around the application software and programmable hardware using one of the Development NanoBoards such as the Desktop NanoBoard. Very little regard is given to the hardware platform to be used in the final implementation and work can rapidly proceed on proving out and cementing the features of the design. The decision of how (or if) to deploy the newly created system is independent of this level of design.

Assuming you intend to deploy your design beyond one of the Desktop NanoBoard products, you have two degrees of freedom.

1. Hardware Platform – Will you use off-the-shelf (OTS) hardware, create your own, or use a mixture of the two?
2. Enclosure – Will you use an OTS enclosure, create your own from scratch, or modify an existing one?

The following sections discuss how you might work within these degrees of freedom to varying levels.



### Level 2: OTS Hardware Platform, OTS Enclosure

This level is the simplest deployment option as it makes use of both an off-the-shelf hardware platform and enclosure. By using one of Altium's deployment NanoBoards, you can mix and match different daughter boards and peripheral boards to produce a customized hardware platform that is tailored to your application. In addition, enclosing the complete NanoBoard in one of Altium's supplied cases will ensure a professional appearance of your product and avoids the logistical headaches associated with manufacturing. Design compatibility ensures you can seamlessly migrate your design from the Desktop NanoBoard to a complete deployment solution.



Deploying your designs in this fashion allows you to focus primarily on the device intelligence without being bogged down by hardware implementation issues. The unique identification system that has been built into the NanoBoard enables it to probe all connected daughter and peripheral boards and quickly reconfigure the entire design. You can be up and running on your deployment platform in little more than the time it takes to rerun the FPGA build flow.

### **Level 3: OTS Hardware Platform, Custom Enclosure**

Deploying your design using an OTS deployment NanoBoard inside an enclosure of your own design is an incremental step from level 2 that can have a dramatic impact on the level of professionalism that you are able to portray to your customers. Use one of Altium's mechanical STEP models as the basis for customization or construct a completely new enclosure of your own design. Either way, you'll have the ability to tailor the form and fit of your end product to ensure it fits snuggly into its final environment.

Altium Designer's 3D bodies allow you to quickly visualize your end product and trap any interference issues that may crop up between the ECAD and MCAD environments.



### **Level 4: Mixture of OTS and Custom Hardware Platform, OTS or Custom Enclosure**

While Altium is continuously developing more peripheral boards, there still might be occasions when you need to include your own custom hardware as part of the design. The expandability of the NanoBoards ensures that this is a relatively simple task and it gives you the best of both worlds. You can selectively customize the hardware platform while still leveraging off the existing infrastructure that has been designed into the NanoBoard architecture.

As with the previous level, the extent to which you use an existing enclosure or create/customize your own is completely up to you.

### **Level 5: Custom Hardware Platform, OTS or Custom Enclosure**

This final level requires the greatest amount of hardware development but gives you the greatest flexibility in terms of form and fit. In particularly cost-conscious applications it may be necessary to rationalize NanoBoard circuitry to only those subsystems that are absolutely necessary to the design. The design reuse capabilities of Altium Designer makes this process a very quick and easy task. Because all of the NanoBoard circuits are included as design reuse blocks and installed as part of Altium Designer, you can link those blocks into your custom hardware design and avoid having to reinvent the wheel. Altium Designer even includes the part numbers and supplier information of all parts used in the NanoBoards. This makes the process of procuring parts an absolute breeze.

## **Transferring the Design to a Deployment Platform**

---

The deployment level you choose will have some bearing on how simply you can retarget your design. The NanoBoard infrastructure includes intelligence that allows it to probe connected daughter boards and peripheral boards and automatically create a new configuration based on the connected hardware. All hardware supplied by Altium conforms to this standard but if you are using hardware from a third party or your own custom hardware that does not include this feature then you may need to perform some configuration steps manually to arrive at the same destination.

There is plenty of help information already available within Altium Designer to assist with the creation of new constraint files and configurations and so I won't repeat that content here. But it is sufficient to say that once the new configuration has been defined, you can be up and running on your deployment platform in little more than the time it takes to rerun the FPGA build flow.

## **Why isn't the Video Image Spinning?**

---

Throughout the development of this tutorial we have used project names such as SpinningVideo and VideoSpin yet the image that appears on the TFT display remains stationary. The reason for this is because what we have shown you is really just the beginning of what is possible in Altium Designer. If you're keen to develop this project to include rapid zoom and rotation of the video image, then have a look at the Examples\NB2DSK1 Examples\CHC Image Rotation folder in the Altium Designer installation. While you're there, you might notice that this example makes use of the new C-to-Hardware capabilities in Altium Designer.... But that'll have to wait until another tutorial.

## **Revision History**

---

Date	Version No.	Revision
25-Sep-2007	1.0	Initial Release
21-Jan-2008	1.1	Added section on deployment
20-Mar-2008	2.0	Updated for Altium Designer Summer 08

Software, hardware, documentation and related materials:

Copyright © 2008 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.