Wishbone Register bank Interconnect Multi-master Multi-slave (Wrimm)

Version 0.1 July 2014

Introduction

Most FPGA projects I work on require control bits for various modules to configure their operation, monitor their status, or activate their functions.

This always seems an ugly piece of the architecture. The rest of the design is nicely organized in self contained units that are easy to chart in a simple block diagram. Then you have this control structure that infests the entire design.

Developing and maintaing the register map is also a problem if the addresses of the registers are spread all over the source code.

Re-developing an maintaining this same simple functionality again and for each project seemed a large waste of time. Adding a new register to any of the projects frequently required modifying several files and intimate knowledge of the operation of those files.

Over many years I have tried many solutions to this problem:

- Initially I designed a new control system for each block. Obviously this quickly becomes unmanageable.
- Then I discovered Wishbone which provided a unified interface for all the blocks. Now I didn't have to remember the specific interface for each block. I could rely on the Wishbone spec to set the standard for all interfaces.
- In some designs all the blocks would have their own Wishbone interface, and I would have to organize the address space by modifying code in each block. Then I moved to address selection inputs on the instantiation of each block.
- In other designs I would write a custom Wishbone slave just to provide all the register bits required by the project. Then I could just connect register bits from this block to the other blocks that didn't have to understand Wishbone.
- Next I moved to generic Wishbone slaves that provided banks of specific register types. Then I didn't have to modify the code every time I needed an additional register. This approach is workable, but ultimately not very flexible.

All these approaches seemed sub-optimal for different reasons.

The initial goals of Wrimm were:

- Provide register functionality to the system blocks.
- Minimize coding necessary to add, remove, and change the specific registers required for a project.
- Efficiently and concisely manage the address space of the Wishbone bus, so address can be organized sensibly, new address can be allocated, and the address map can be documented without extensive research into source code.
- Provide a stable reliable tested base design that does not require modification for each new project.

To achieve these goals I researched the VHDL language and learned some new techniques. The code of Wrimm may be unlike the code you are used to. It is different from most of the other project code I have seen. On the other hand it is all synthesize-able VHDL.

An end result of this design methodology is a large reliance on synthesis optimization. Many functions provided in the design of Wrimm may not be used in every instance. In this case the synthesis tool is expected to optimize them away. Large blocks of logic are specified, then handed to the tool for optimization. If you are uncomfortable with this approach you will likely be uncomfortable with this project.

On the other hand, using these techniques enabled several other goals:

- Multi-master arbitrated interconnect functionality
- Interconnect functionality to support multiple slaves with partial address decoding.

So far my results have provided the fastest, smallest register interface designs I have ever seen. YMMV.

Source Code Structure & Customization Strategy

The Wrimm interconnect and register functionality is contained in a single VHDL entity in Wrimm.vhd. This code should not require modification from project to project.

The Wrimm entity relies on a package file, WrimmPackage. This file contains constant and custom type declarations which must be modified to suit each project.

Take the port list of the Wrimm entity as an example. Nearly all the ports listed are of custom types. These custom types are all declared in the WrimmPackage. This allows the SettingRegs output of the Wrimm entity to provide what ever number of setting registers is declared in WrimmPackage.

The WrimmPackage included in this project only contains example register and slave parameters. Your new project will require changes to the constants declared in WrimmPackage. If the changes follow the format of the example registers, you should not need to make any changes to the Wrimm entity or architecture to support the custom features of your design.

Take a look at the implementation in the top level example Wrimm_Top and see if you like the look of the result. Hopefully you will immediately see how easy it will be to add and change registers as your design matures.

Wishbone Interface (Wishbone Datasheet)

Wrimm provides multiple slave and master interfaces. The number of slave and master interfaces is defined in the WrimmPackage for each unique project.

To provide ports for n Wishbone interfaces, the Wrimm port list declares an array of Wishbone port interfaces. This way any number of Wishbone masters and slaves may be supported without changing the entity declaration or architecture implementation.

To support these arrays, WrimmPackage declares two VDHL record types: WbMasterOutType and WbSlaveOutType. These two types contain all the required and supported optional Wishbone signals (except clock and reset which are global).

Wishbone Datasheet Specs

General Description	Wishbone Register Bank Interconnect Multi-master Multi-Slave
Wishbone Version	B4
Type of Interface	Slave and Multi-master Interconnect
Supported cycles	Slave, Read/Write Slave, Block Read/Write Slave, RMW
ERR_I handling	ERR_I passed on to master
ERR_O handling	ERR_O returned for strobes to unrecognized addresses
RTY_I handling	RTY_I passed on to master
RTY_O handling	RTY_O never generated
Tags	currently no tags supported
Data Port Size	Port size configurable with constant declaration
Data Port granularity	currently no sub port granularity supported
Data Port Operand size	Operand size is the same as the declared port size
Data transfer ordering	Big/Little Endian (port size=granularity)
Data transfer sequencing	Undefined?
Clock Constraints	Frequency dependent on implementation

Slave Interface(s) (Connected to Masters)

Signal Name	WISHBONE Equiv.
WbClk	CLK_I

Signal Name	WISHBONE Equiv.
WbRst	RST_I
WbMasterIn(x).Strobe	STB_I
WbMasterIn(x).WrEn	WE_I
WbMasterIn(x).Addr	ADR_I(0..n)
WbMasterIn(x).Data	DAT_I(0..n)
WbMasterIn(x).Cyc	CYC_I
WbMasterOut(x).Ack	ACK_O
WbMasterOut(x).Err	ERR_O
WbMasterOut(x).Rty	RTY_O
WbMasterOut(x).Data	DAT_O(0..n)

Master Interface(s) (Connected to Slaves)

Signal Name	WISHBONE Equiv.
WbClk	CLK_I
WbRst	RST_I
WbSlaveOut(y).Strobe	STB_I
WbSlaveOut(y).WrEn	WE_I
WbSlaveOut(y).Addr	ADR_I(0..n)
WbSlaveOut(y).Data	DAT_I(0..n)
WbSlaveOut(y).Cyc	CYC_I
WbSlaveOut(y).Ack	ACK_O
WbSlaveIn(y).Err	ERR_O
WbSlaveIn(y).Rty	RTY_O
WbSlaveIn(y).Data	DAT_O(0..n)

Register Interface

Wrimm provides three different register interfaces. These three type have proven through experience to satisfy many register interface requirements. All three provide a data interface between a controller and a module. The controller is a Wishbone master, in turn that Wishbone master may be driven by some larger control system on chip or off chip. The module is the stupid function on the other end of the register interface that communicates through these dedicated signals.

1. Status registers allow controllers to monitor outputs from a module.
2. Setting registers allow controllers to configure module parameters.
3. Trigger registers allow controllers to initiate module activities.

Wrimm supports any number of each of these register types. Each of these register types is configured differently in WrimmPackage and used differently by the system.

Status Registers

Configuration

Status registers require a unique name for each register. These names are declared in the StatusFieldType declaration in WrimmPackage. StatusA, StatusB, and StatusC are the status registers declared in the example WrimmPackage.

Each status register can be any length from 1 to the data port width declared by WbDataBits. The length of each register is defined with the BitWidth field of the StatusParams constant.

If the status register is less than WbDataBits, you can define its position in the logical register. This location is defined with the MSBLoc field of the StatusParams constant. Bits are big endian ordered so to left justify the field set MSBLoc => 0. The maximum MSBLoc is (WbDataBits- Status register BitWidth).

The Wishbone bus address of each status register must be defined with the Address field. The length of the address field must match the width of the Wishbone address bus which is defined by the WbAddrBits constant.

Interface

Status registers simply require connecting the output bits from the module to the register input created by the status register definition. This can be done in the Wrimm incantation port map.

StatusRegs(StatusA) => ModuleAStatusBits,

Setting Registers

Configuration

Setting registers require a unique name for each register. These names are declared in the StatusFieldType declaration in WrimmPackage. SettingX, SettingY, and SettingZ are the setting registers declared in the example WrimmPackage.

Each Setting register can be any length from 1 to the data port width declared by WbDataBits. The length of each register is defined with the BitWidth field of the StatusParams constant.

If the setting register is less than WbDataBits, you can define its position in the logical register. This location is defined with the MSBLoc field of the SettingsParams constant. Bits are big endian ordered so to left justify the field set MSBLoc => 0. The maximum MSBLoc is (WbDataBits- Setting register BitWidth).

The Wishbone bus address of each status register must be defined with the Address field. The length of the address field must match the width of the Wishbone address bus which is defined by the WbAddrBits constant.

Setting registers can also be configured with a default value specified in the Default field. The default value must be WbDataBits wide (not the defined BitWidth). If BitWidth is less than WbData bits, the default should be right aligned regardless of MSBLoc.

Interface

Setting registers each have two interface ports.

The first is an output port that provides the setting data to the module. The output port is WbDataBits wide. Registers with BitWidth less than WbDataBits will provide the data right aligned in the output port.

This port can be assigned in the port map of the Wrimm instantiation, or the port can be mapped to a SettingArray signal, and each register mapped to its destination by selecting the array element from that signal

```
SettingRegs(SettingX)       => ModuleXSettingReg,
or

SettingRegs                 => settingRegsSignal,
and
moduleXSetting      <= settingRegsSignal(SettingX);
```

Setting registers also provide an input to synchronously reset the register to its default value.

These reset bits are connected to the SettingRsts(RegName) ports.

Trigger Registers

Configuration

Trigger registers require a unique name for each register. These names are declared in the TriggerFieldType declaration in WrimmPackage. TriggerR, TriggerS, and TriggerT are the trigger registers declared in the example WrimmPackage.

All trigger registers a 1 bit wide so their width does not need defined for each instance.

The position of the trigger bit in the logical register is defined with the MSBLoc field of the TriggerParams constant. Bits are big endian ordered so to left justify the field set MSBLoc => 0. The maximum MSBLoc is WbDataBits-1.

The Wishbone bus address of each trigger register must be defined with the Address field. The length of the address field must match the width of the Wishbone address bus which is defined by the WbAddrBits constant.

Interface

Trigger registers require connecting trigger output bit to the module This can be done with the same methods used by setting registers. The only difference is all trigger registers are one bit wide.

Trigger registers also provide a synchronous clear signal, port TriggerClr(TriggerName). This provides extended trigger pulses for modules operating on clocks slower than the Wishbone clock. If a single cycle trigger pulse is sufficient, the trigger clear can be tied high. The trigger register may be read by the master to see if the module has cleared it yet.

Acknowledgements