The Butterfly Grid Computer

Table of Contents

[Preface 2](#_Toc486068034)

[Who this book is for 2](#_Toc486068035)

[Motivation 2](#_Toc486068036)

[What this book is about 2](#_Toc486068037)

[About the Author: 2](#_Toc486068038)

[Book Organization 2](#_Toc486068039)

[Hardware 3](#_Toc486068040)

[The SoC 3](#_Toc486068041)

[The Grid 4](#_Toc486068042)

[Organization 4](#_Toc486068043)

[Block Diagram of Grid 4](#_Toc486068044)

[A Node 5](#_Toc486068045)

[Node $111 6](#_Toc486068046)

[Node $211 6](#_Toc486068047)

[Router 6](#_Toc486068048)

[Structure 6](#_Toc486068049)

[CPU 7](#_Toc486068050)

[System Clocks 8](#_Toc486068051)

[Networking Standard 8](#_Toc486068052)

[Overview 8](#_Toc486068053)

[Messages 8](#_Toc486068054)

[Inter-grid Connections 10](#_Toc486068055)

[TMDS 10](#_Toc486068056)

[Signal Synchronization (or lack thereof) 10](#_Toc486068057)

[Software 12](#_Toc486068058)

[The Operating System 12](#_Toc486068059)

[Workload Distribution 12](#_Toc486068060)

# Preface

## Who this book is for

This book is for the FPGA enthusiast who’s interested in the basics of grid computing. It’s advisable that one have a fairly good background in digital electronics, computer systems and networking before attempting a read. Examples are provided in the Verilog language, it would be helpful to have some understanding of HDL languages. If you’re into electronics and computers as a hobby FPGA’s can be a lot of fun. The book attempts to be ‘hands-on’ in nature and provides sample program code.

## Motivation

The author has never built a grid computer before and thought it might be interesting to do so. The author was also interested in building a fault tolerant computer and that required redundancies. Large arrays of computing elements represent the current state of the art. An example of such a computer would be Google’s Tensor processing array. Much processing these days is done with large arrays of parallel computers. Having a deep understanding of how they work is very satisfying. In short this is a learning exercise for the author.

## What this book is about

This book is about the Butterfly Grid Computer. A grid computer is a form of parallel processing where processing nodes are organized into a grid shape.

## About the Author:

First a warning: I’m an enthusiastic hobbyist like yourself, with a ton of experience. I’ve spent a lot of time at home doing research and implementing several soft-core processors, almost maniacally. One of the first cores I worked on was a 6502 emulation. I then went on to develop the Butterfly32 core. Later the Raptor64. I have about 20 years professional experience working on banking applications at a variety of language levels including assembler. So I have some real world experience developing complex applications. I also have a diploma in electronics engineering technology. Some of the cores I work on these days are really too complex and too large to do at home on an inexpensive FPGA. I await bigger, better, faster boards yet to come.

## Book Organization

This book is organized into two sections, one for hardware and one for software.

# Hardware

## The SoC

The Butterfly system-on-chip contains a grid computer along with a number of peripheral devices. Peripheral devices include text video output, keyboard input, real-time clock interface.

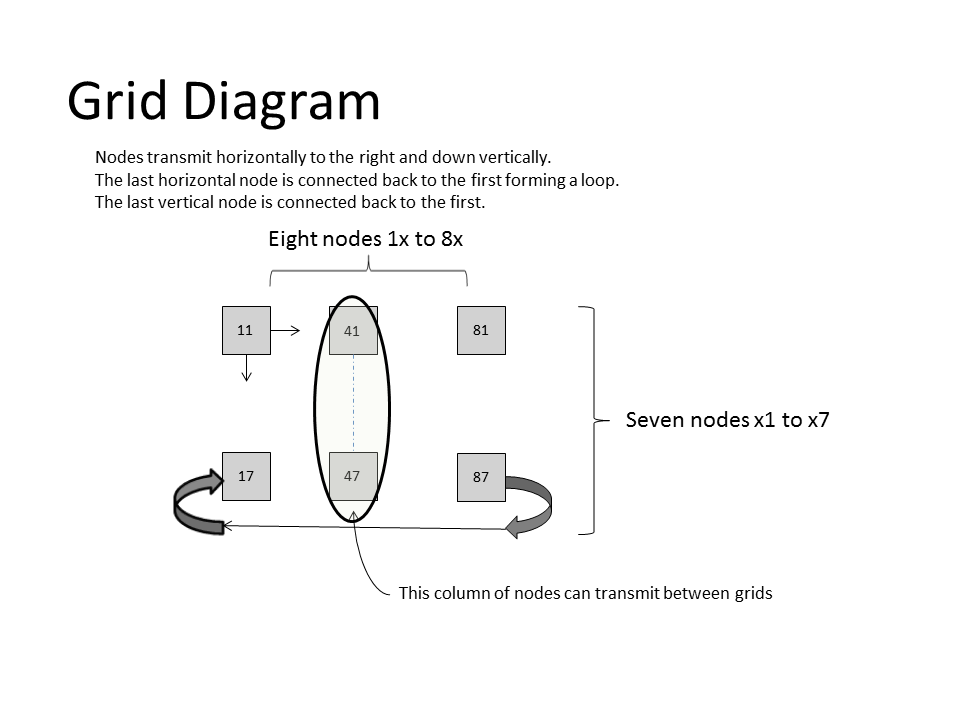
## The Grid

### Organization

The grid computer is organized as an X, Y plane of nodes with a backplane capable of connecting multiple grids together. The grid isn’t a pure 3D mesh. It’s organized more like a shelving unit. Connection of multiple grids has a wiring limitation. A single grid is located entirely on the FPGA, additional grids would be located in additional external FPGA’s. It would be required to transmit the Z channel for all the nodes in the network in order to achieve a 3D mesh. This could potentially be a lot of channels. The trade-off in connecting between grids is to use fewer higher speed channels rather than many low speed ones. Fewer high speed channels require fewer hardware resources than would be required for many low speed channels.

The grid is currently an eight by seven array of processing nodes. This organization was chosen as what would fit easily into the FPGA device. Originally the grid was an 8 x 8 array but as functionality was added resources available in the FPGA became too sparse and the array wouldn’t place and route.

### Block Diagram of Grid

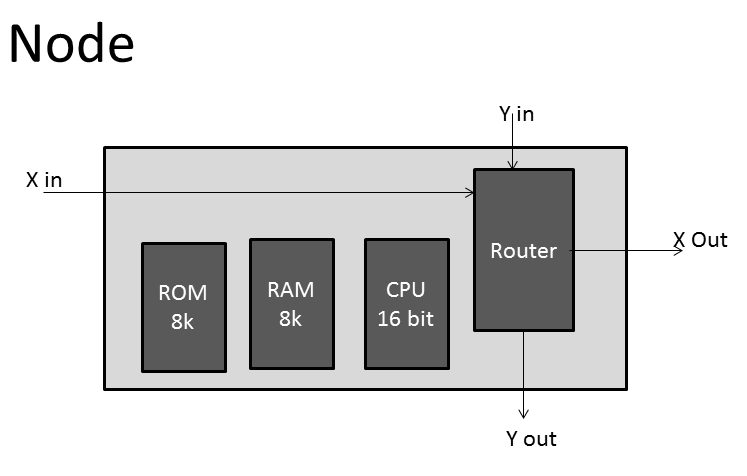


## A Node

For an on-chip network the network nodes needs to be small and simple. It would not be possible to have a substantial network on chip with large complex components. As such the CPU used (Butterfly16) is a small (about 500LUTs) 16 bit cpu. Each node has only a small RAM (8kB) and ROM (8kb) associated with it. The node’s router is probably the largest component of the node.

Each node is given a 12 bit ID that is of the form {X,Y,Z}. Four bits represent each of the X, Y and Z positions within the network. The maximum number of nodes in the network is then 4096.

Block Diagram of Typical Node



Each node has very few external connections. The nodes are connected together via four bit parallel signals. Other than the UART connection signal, reset, and clock most nodes don’t have any other connections.

Verilog code for typical node connection:

|  |
| --- |
| node #({8'h71,Z}) un71  (  .rst\_i(rst\_i),  .clk\_i(clk\_i),  .sclk(clk57),  .rxdX(txdX61),  .txdX(txdX71),  .rxdY(txdY77),  .txdY(txdY71)  ); |

Memory is not directly shared between nodes.

## Node $111

Grid node number $111 is designated as the master node. It contains the master version of the Tiny BASIC interpreter. It is also connected to the text video display and LEDs. Other nodes requiring text output must request it from this node.

## Node $211

Node $211 acts as an input device processor and is connected to the keyboard, push buttons, and switches. Other nodes requiring input must request it from this node.

## Router

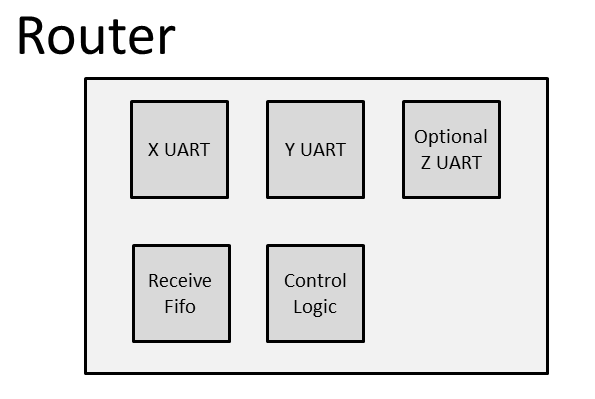
### Structure

The router contains potentially three UARTs depending on where it’s located. The centre column of nodes (addresses begin with $4xx) is capable of routing in the Z direction and so contains a third UART in the router. The other routers contain only two UARTs in order to minimize the logic resources used.

Internally the router polls all the UARTs for received messages and re-transmits them on the network if required. The router also polls for a request by the node to transmit a message. Receiving messages takes priority over transmitting messages.

In the router receivers make use of receive fifos, transmitting does not make use of a fifo in order to minimize hardware resources required. Instead a transmit busy bit is polled to know when it’s okay to transmit.

Each UART receiver has a 32 entry fifo for received messages. This fifo is used to transfer all network messages. There is an additional 32 entry fifo dedicated to messages received specific to the node.



## CPU

The cpu used for all the nodes is the Butterfly16 cpu which is a 16 bit processor. This processor was chosen for it’s small size (500LUTs). It’s not a very powerful processor, but there a lot of them in the grid. It does support a fairly standard instruction set and code density is reasonable. Some of the basic operations include: add, subtract, bitwise and, or, and exclusive or, shifts and rotates by one bit to the left or right, load and stores of bytes or 16 bit words.

## System Clocks

The system makes use of a number of different clocks. The clocks represent trade-offs in hardware. Most of the clocks were chosen for simplicity of generation. Some of the clocks are required to be specific frequencies. For example the DDR3 rate generation clock of 200MHz. Because some of the clock frequencies are fixed other clocks must be submultiples. The module generating the primary clocks uses only a single MMCM component. The clocks and usage are listed below.

|  |  |  |
| --- | --- | --- |
| Rate (MHz) | Divisor | Description |
| 100 |  | external oscillator input multiplied by 12 to generate the primary system clock |
| 1200 | 1 | base system clock, all other clocks are derived from this one. This clock was chosen as being near the upper limit for the FPGA device and was also chosen to make it easy to generate DDR3 and TMDS clocks. |
| 400 | 3 | timing clock for video output, and also timing clock for inter-grid communications, the DDR controller also re-generates this clock rate. |
| 200 | 6 | reference clock, also used for DDR3 reference clock |
| 80 | 15 | video pixel clock, reference for 1280x768 display. This clock must be 1/5 the 400 MHz rate. |
| 57 | 21 | node operating clock, and network communication clock. This clock must 1/7 th the 400 MHz clock for the TMDS signalling between grids. |

## Networking Standard

### Overview

The networking standard is a custom network; it’s not a standard that anybody I know of uses. It’s designed to be just about as simple as possible. Ethernet it ain’t. The network standard features fixed length messages and fixed routing directions. Data is transferred at a constant clock rate (57 MHz) in a four bits parallel UART fashion using a 4x baud clock. The resulting transmission rate is 57Mb/s.

### Messages

Fixed length messages are passed around the network. A fixed length message size of 128 bits was chosen in order to keep the routers simple. There are a number of different types of messages identified by the message type field in the message that may be used. The message size is a trade-off in the amount of hardware required for the router.

Message Format

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DST XY | SRC XY | reserved | CTL | TTL | TYPE | DZ | SZ | Payload |
| 8 | 8 | 32 | 2 | 6 | 8 | 4 | 4 | 56 |

|  |  |
| --- | --- |
| Field | Description |
| DST XY | the X,Y co-ordinates of the target (destination) node |
| SRC XY | the X,Y co-ordinates of the source node |
| reserved | this field is reserved for routing information |
| CTL | this field is reserved for control information and must be zero |
| TTL | Time to live. This field indicates the number of network hops the message will travel before being automatically removed from the network. It’s present to prevent network congestion if a destination node isn’t responding. |
| TYPE | This field indicates the type of message. Message types include reset, ping, stop, run program, request input, etc. Message types are found in MessageTypes.asm |
| DZ | This field contains the Z co-ordinate for the destination node |
| SZ | This field contains the Z co-ordinate for the source node. |
| PAYLOAD | This field contains information pertaining to the message. |

The router processes message as 136 bit packets with a leading zero nybble and a trailing nybble of all ones. Using a leading zero nybble and trailing one’s packet allows the router to determine if the message is properly framed.

Messages always route to the right (+X) and down (+Y). In the case of a grid to grid message requiring a Z direction messages route to the right until they hit a node with a Z UART then they route along the Z direction until the proper grid is reached. Determining how to route a message is simple and is based on the target node’s ID. ID’s are assigned to nodes based on their location in the grid. This is unlike the internet where ID’s are somewhat arbitrary.

When a message is sent the router first tests to see if the X co-ordinate of the router matches the X co-ordinate of the target node. If so then then the Y co-ordinate is tested. If all three co-ordinates match the router’s co-ordinates then the message is placed into the receive fifo for the node.

### Inter-grid Connections

Inter-grid connections between FPGA boards make use of HDMI ports and their TMDS transceivers. This means was chosen because it’s high speed and makes use of resources available on the FPGA board. Cabling is a simple HDMI cable.

### TMDS

TMDS stands for Transition Minimized Differential Signal. TMDS is used by the DVI and HDMI video interfaces. TMDS itself is not limited to those interfaces; it is also used in other communications such as it is used here. The encoding algorithm allows for reliable clock recovery and reduces EMI produced by the transmission cables.

The TMDS standard also attempts to maintain a stable dc bias on the signal lines. For HMDI signals an 8b/10b encoding is used with four specially encoded control tokens. A TMDS signalling standard is used for the Butterfly NoC however it is not the HDMI standard. For the Butterfly NoC a 12b/14b encoding is used to make maximum use of available hardware. This works in much the same way as an 8b/10b encoding but is used to encode three four bit parallel channels onto each signal wire pair.

The TMDS encoder is a hacked up version of the TMDS\_encoder.vhd provided by Digilient Incorporated. Following the philosophy of “don’t do all the work yourself” this encoder was modified to encode a 12b/14b signal, and the control token logic was removed. Butterfly NoC does not require the HDMI control tokens used for synchronization and phase alignment of signals.

### Signal Synchronization (or lack thereof)

High-speed serial communications can get to be very complex. Because signals are travelling down a length of wire signals on different wires can become slightly out of phase with each other. This presents a problem when different channels encoded on the signal wires need to be used in sync with each other. A means to synchronize all the signals at the receiving end is required. If the reader is interested in finding out what is required to re-synchronize signals in a receiver he is encouraged to study HDMI receiver logic, it’s beyond the scope of this book. The Butterfly network gets past the requirement of synchronized signals by using a UART style asynchronous transmission format and keeping the entire contents of a channel on a single wire pair rather than distributing it across multiple wires. Travelling down the wire different channels may indeed become out of sync. But since the channels are used independently it doesn’t matter.

# Software

## The Operating System

The grid is inherently multi-tasking. It is message based. Each node in the grid is effectively a thread of execution. To keep things simple each grid node only supports a single thread of execution. With an 8x7 grid that’s 56 parallel threads allowed. There is no multi-tasking kernel; it isn’t required. There is no swapping of task state required because the entire state can be contained in a node. The grid works by having nodes sit in a loop receiving messages similar to a message loop for a gui operating system. As a node receives messages it invokes message handlers to process the message and potentially transmit data back to the network. The amount of software that is located on the node is very limited, so the software is organized in a simple fashion.

|  |
| --- |
| Start:  poll for message receipt  invoke message handler  go back to Start: |

Each node at a minimum requires driver software for the node’s router and the ability to download additional software and data as required.

## Workload Distribution

One of the great things about a grid computer is that processing workload can be distributed across multiple nodes. Some of the grids workload is statically allocated based on the hardware connected to a node. Most nodes have extremely limited hardware resources consisting of only ram, rom, cpu, and router. There are some nodes however that have extra resources attached to them. The grid computer wouldn’t be very useful if there wasn’t any input / output devices attached to it.