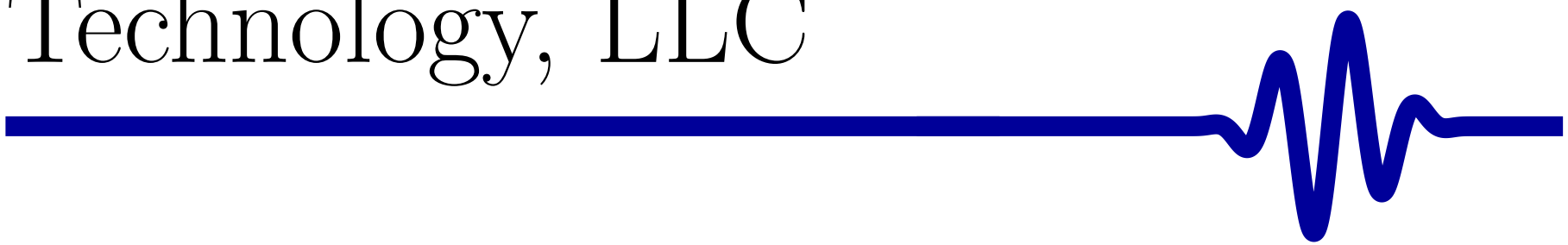
AutoFPGA

An FPGA Component Aggregator

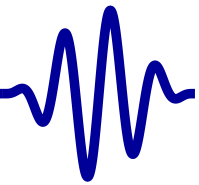
Gisselquist Technology, LLC

Daniel E. Gisselquist, Ph.D.

September, 2017



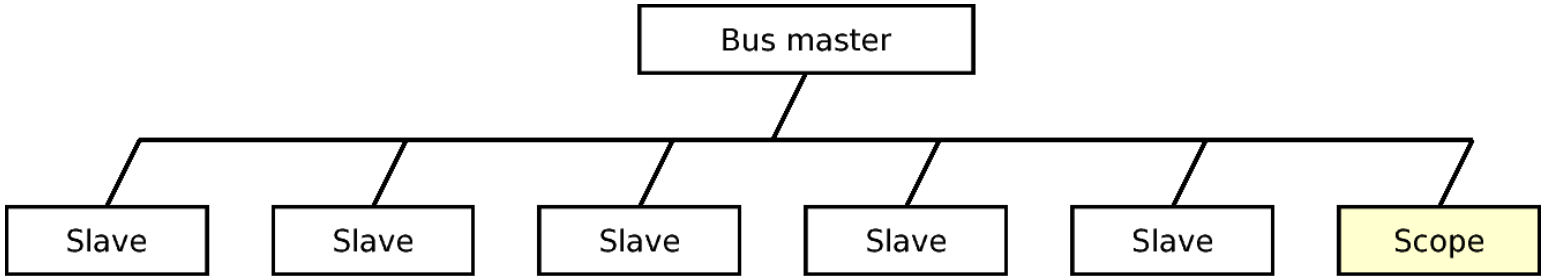
Overview



- Why AutoFPGA?
- Copy/Paste
- Data Structure
- Enhancements to the basic simplified ZipCPU
- What performance can be expected?

A Basic Bus

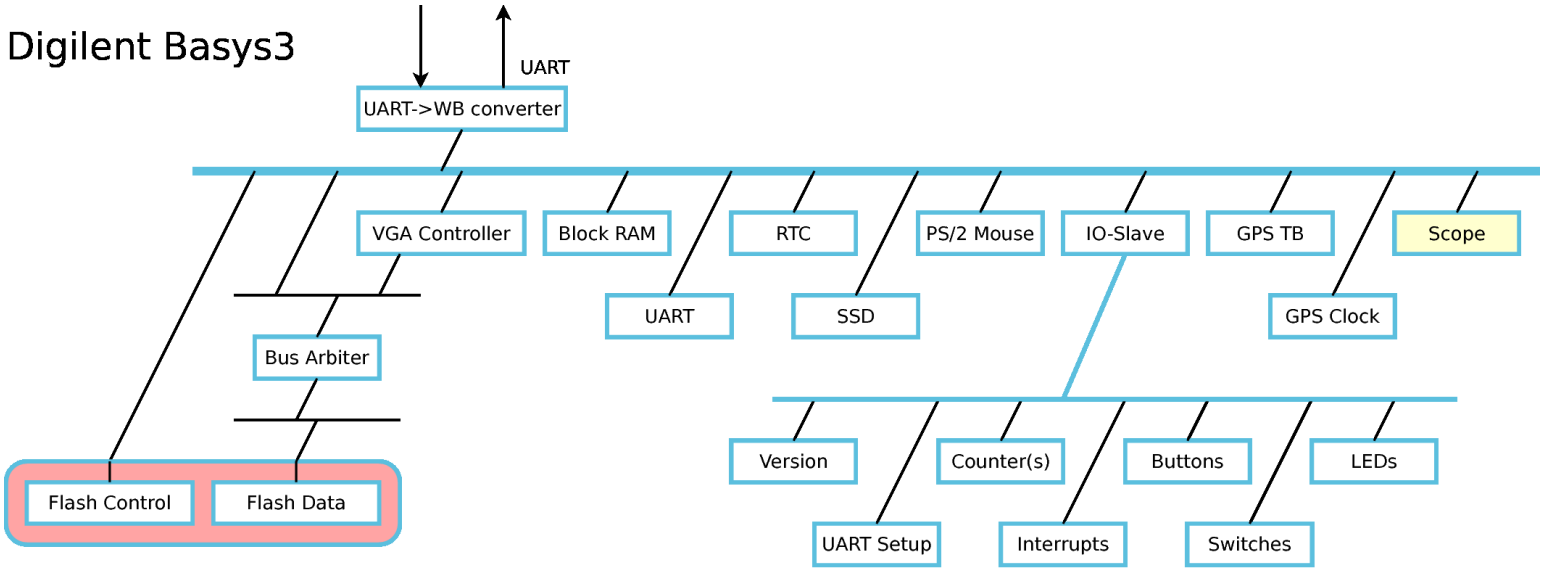
After building several designs, they all started to fell like they had the same bus structure ...



Let's take a quick tour of some of those designs.

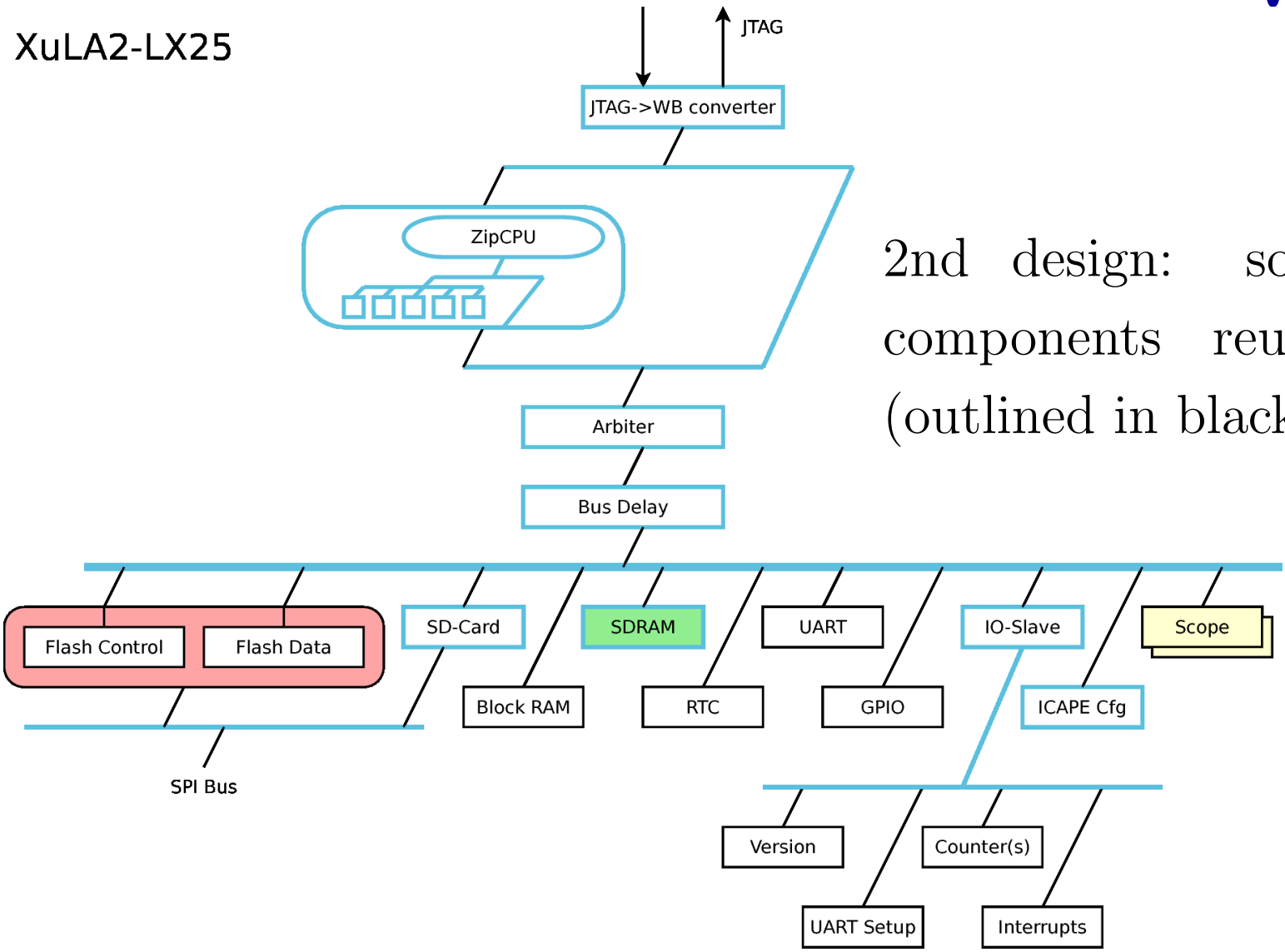
Tour: Basys3

Here was the bus structure for my first open design



New components/work are outlined in blue

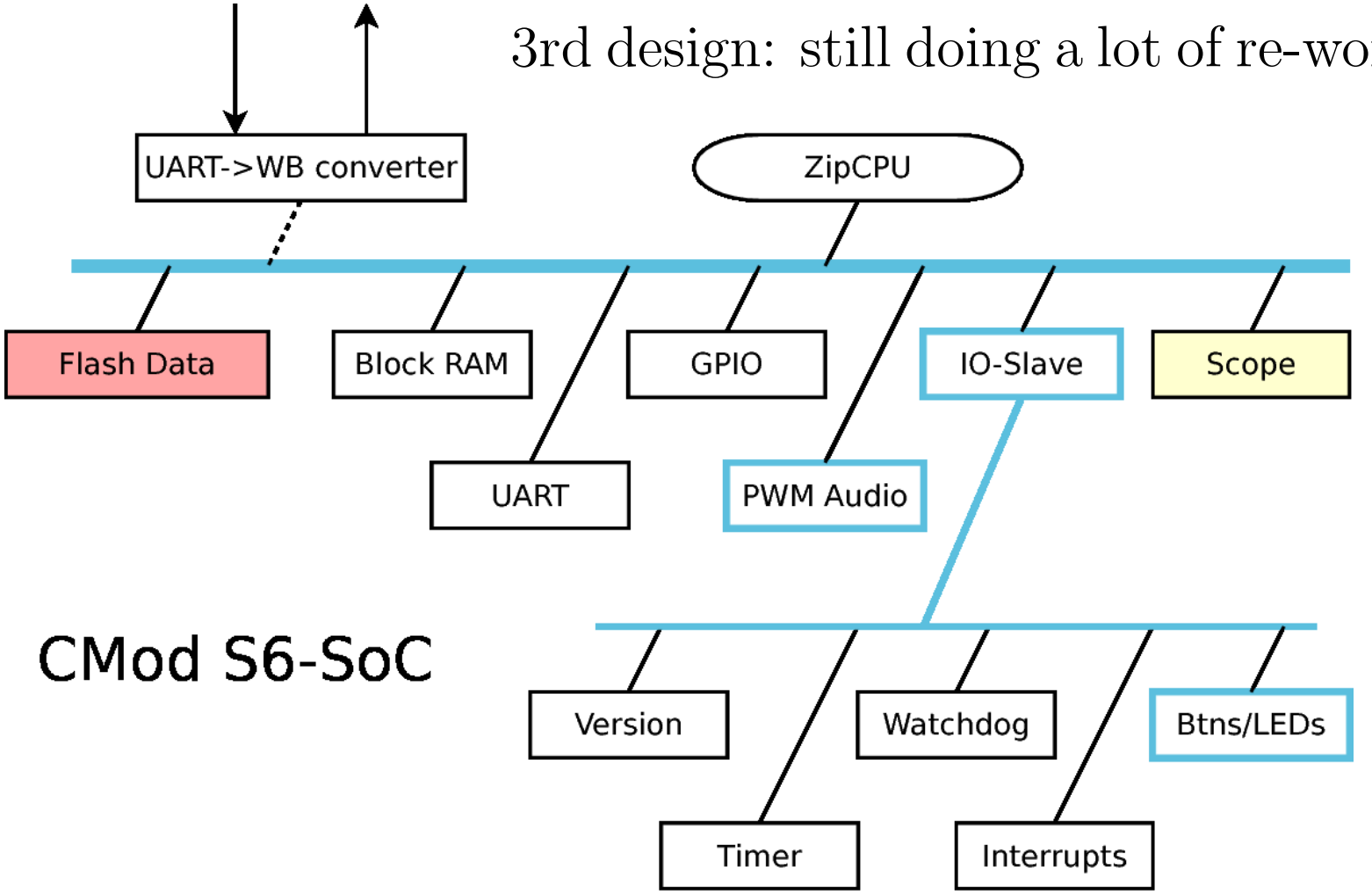
Tour: XuLA2



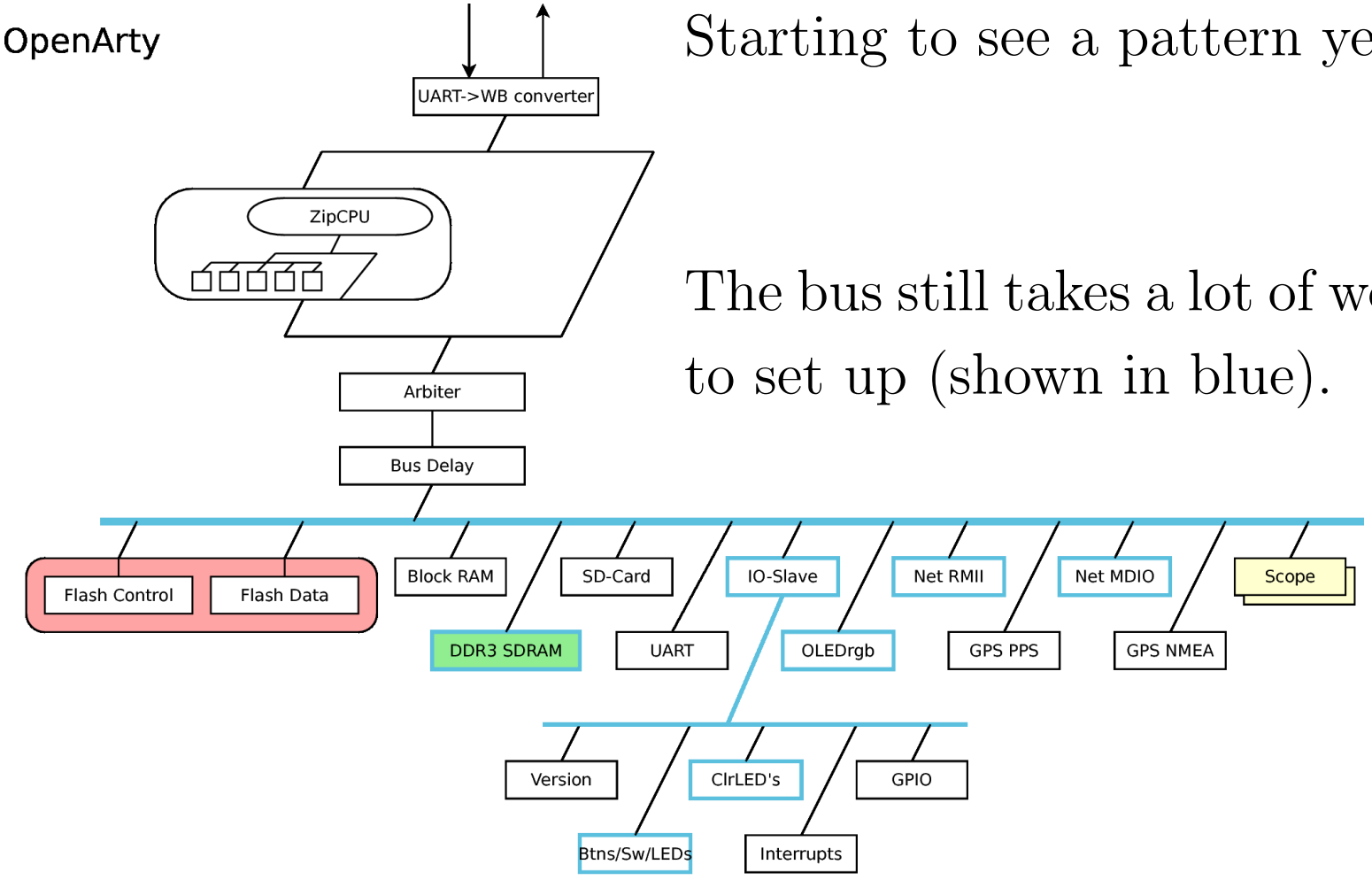
2nd design: some components reused (outlined in black).

Tour: S6, LX4

3rd design: still doing a lot of re-work.



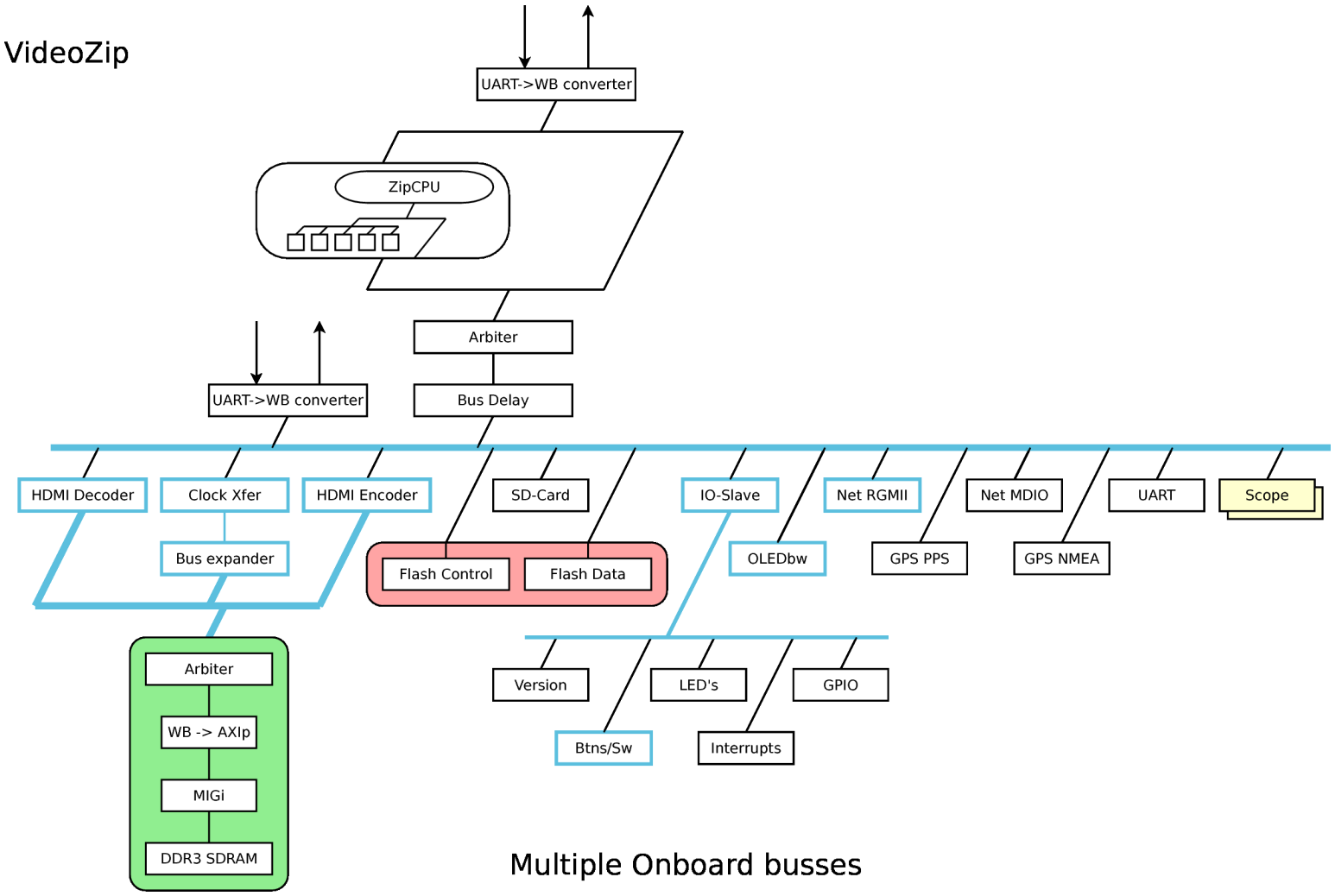
Tour: Arty



Starting to see a pattern yet?

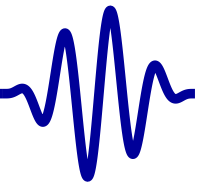
The bus still takes a lot of work to set up (shown in blue).

Tour: NexysVideo



The interconnect remains the majority of the work

Lessons Learned



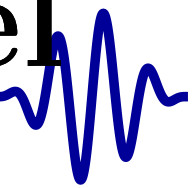
Every design had a lot in common

- Lots of components were reused
  - UART, block RAM, flash, Real-Time Clock, GPIO, SD-card, WB-Scope etc.
- Bus masters were reused
  - ZipCPU, UART based debugging bus

Putting those components together necessitated lots of work every time!

- Discourages ad-hoc components

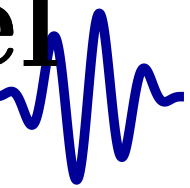
Rebuilding the Wheel



The *interconnect* was rebuilt with every new design

- New address assignment
  - Adding in debugging registers
  - Block RAM re-allocation?
- New interrupt assignment
- Clock sensitive timing parameters all change
  - UART, real-time-Clock, GPS clock tracking core, (and more) *all* require knowing the speed of the clock

Rebuilding the Wheel



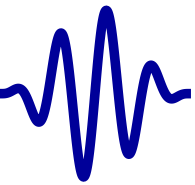
Any change necessitated a change to:

- Integrated full-design simulator
- FPGA Constraint file (UCF/XDC/etc)
- Header files for the ZipCPU
- libGloss: newlib's per CPU file
- Register name file(s)
- Design documentation

It was a mess!

Did I mention that this discourages ad-hoc design changes?

Proprietary Soln's

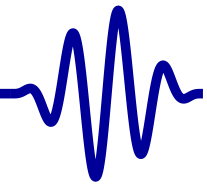


Many vendors offer proprietary solutions to these problems

- Key design details are hidden and immutable
- Impossible to debug, trace through, or adjust timing
- Tie toolchain updates to the component aggregator
  - Updating the FPGA toolchain can break all tool-chain defined components

"*Closed source soft CPUs are the worst of two worlds. You have to worry about resource use and timing without being able to analyze it.*"

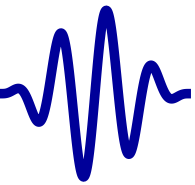
Open Soln's



Open Solutions:

- MiGen is a new language (Python)
- Existing code needs to be ported to MiGen's python
- Rewriting code in order to use the tool defeats the purpose of code reuse

AutoFPGA goals



AutoFPGA: a *simpler*, *open* component aggregator

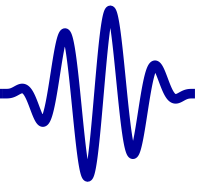
- Given a list of component files, build a design
  - (Roughly) One config file per component
  - Component files should be reconfigurable,
  - Human readable/editable, and
  - No more complex than the underlying languages
- Create project files

  ...that can be used with or without AutoFPGA
- Hide nothing

  Maintain the look/feel of an OpenSource project
- Preserve comments

  Paste comments from the config files into the design

AutoFPGA



AutoFPGA is . . . a glorified Copy/Paste utility

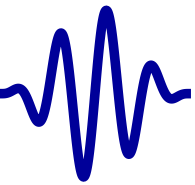
. . . that supports variables that can cross output files

Builds several component files

- RTL / `toplevel.v`, `main.v`
- RTL / `make.inc`, `board.xdc|ucf`
- SIM / `testb.h` (multiple clock sim support)
- SIM / `main_tb.cpp` (Component sim support)
- SW / HOST / `regdefs.h` /`.cpp` (Register naming)
- SW / ZLIB / `board.h`, `board.ld`

Does not build a complete design

AutoFPGA data



Component files consist of key/value pairs

- Basic format is: `@KEY=VALUE`
  - Values are strings, integers, or integer expressions
  - Values can take multiple lines
  - Values are ended by either the end of file, or the next key
- Keys are hierarchical, ex. `@CLOCK.FREQUENCY`
- A `@$` prefix specifies a numeric value, ex `@$KEY=(5+9)/4`
- Substitution: `@$(KEY)` is replaced by its `VALUE`
- `@PREFIX=component_name` is special
  - All keys following will be prefixed w/ `component_name.`
  - Creates a sort of local key scope
  - Terminated by EOF or the next `@PREFIX` key

Looking at main.v

The typical component influences the `main.v` in several places:

- ‘define conditional synthesis (`@ACCESS`, `@DEPENDS`)

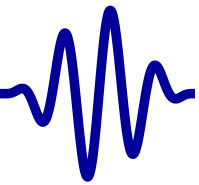
```
‘define @$(ACCESS)
```

- External ports (`@MAIN.PORTLIST`, `@MAIN.IODECL`)
- Signals that need to be defined (`@MAIN.DEFNS`)
- Logic that needs to be inserted

```
‘ifdef @$(ACCESS)
  @$(MAIN.INSERT)
‘else
  @$(MAIN.ALT)
‘endif
```

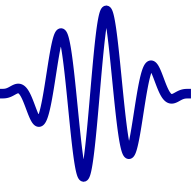
- AutoFPGA based wishbone interconnect logic

Buttons Example



```
@PREFIX=buttons
@NADDR=1
@SLAVE.TYPE=SINGLE
@SLAVE.BUS=wb
@NBUTTONS=5;
@MAIN.PORTLIST=
    // Button inputs
    i_button
@@MAIN.IODECL=
    input   wire   [(@$(NBUTTONS)-1):0]   i_button;
@MAIN.INSERT=
    debouncer #(.NBUTTONS(@$(NBUTTONS)))
          dbi(i_clk, i_button, buttons_data);
```

Buttons Continued



```
@REGS.N=1
@REGS.0=0 R_BUTTONS BUTTONS
@BDEF.OSDEF=_BOARD_HAS_BUTTONS
@BDEF.DEFN=
#define NBUTTONS @$(NBUTTONS)
```

Becomes in `regdefs.h`:

```
#define R_BUTTONS 0xaddress
```

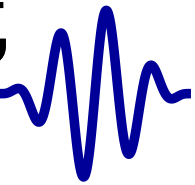
Becomes in `regdefs.cpp`:

```
    { R_BUTTONS, ``BUTTONS'' },
```

Becomes in `board.h`:

```
#define _BOARD_HAS_BUTTONS
```

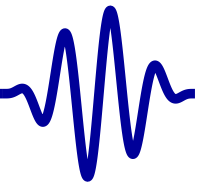
Example component



Example: Include block RAM into your design

```
@PREFIX=bkram
@$LGMEMSZ=20    // Remember: don't repeat yourself
@$NADDR=(1<<(@$(LGMEMSZ)-2))
@SLAVE.TYPE=MEMORY // Declare this in linker scripts
@SLAVE.BUS=wb
@MAIN.INSERT=
  memdev #(.LGMEMSZ(@$(LGMEMSZ)))
    bkrami(i_clk,
        (wb_cyc), (wb_stb)&&(bkram_sel), wb_we,
            wb_addr[(@$(LGMEMSZ)-3):0], wb_data, wb_sel,
        bkram_ack, bkram_stall, bkram_data);
```

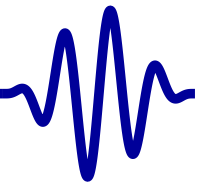
Bus Support



- Bus descriptions include:
  - `@BUS.NAME`, a prefix for all bus wires, ex: `wb`
  - `@BUS.WIDTH`, examples 32 or 128
  - `@BUS.AWID`, calculated internally
  - `@BUS.TYPE`, Currently only WB/B4/pipeline is supported

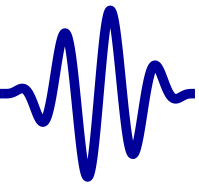
    The code is written in C++. New bus types can be supported by simple inheritance.

Bus Slaves



- Four types of bus slaves (`@SLAVE.TYPE`)
  - `SINGLE`: Single address, pre-known value
  - `DOUBLE`: Multiple addresses, value known one clock after request
  - `OTHER`: May stall the bus and take multiple clocks
  - `MEMORY`: Same as `OTHER`, but w/ linker script support
  - Each slave gets: `@$(PREFIX)_ack`, `@$(PREFIX)_stall`, and `@$(PREFIX)_data` to define and use, as well as `@$(PREFIX)_sel` line.
- Once `@SLAVE.BUS` matches a `@BUS.NAME`
  - `@SLAVE.BUS` is expanded into a full bus description `@SLAVE.BUS.NAME`, `@SLAVE.BUS.WIDTH`, `@SLAVE.BUS.TYPE`, etc.

Bus Slaves



Two other items are important for attaching to this bus:

- `@$NADDR` defines the number of (word) addresses used
- `@$BASE`: The (octet) address on this bus
- `@$REGBASE`: The global (octet) address

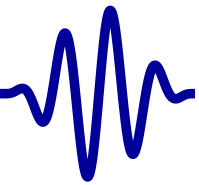
You can then adjust your C-library or Linux board def'n file:

```
@BDEF.IONAME= _component
@BDEF.IOTYPE= unsigned
@BDEF.OSVAL=static volatile @$(BDEF.IOTYPE) *const
  @$(BDEF.IONAME) = ((@$(BDEF.IOTYPE) *)@$(REGBASE));
```

which inserts into `board.h`:

```
static volatile unsigned *const
    _component = ((unsigned *)0x<address>);
```

Bus Masters



- `@MASTER.TYPE` used to specify a bus master
- `@MASTER.BUS` specifies the bus name this component masters
  - Each bus master gets predefined bus wires to use

    `@$(PREFIX)_cyc`, `@$(PREFIX)_stb`,

    `@$(PREFIX)_we`, `@$(PREFIX)_addr`,

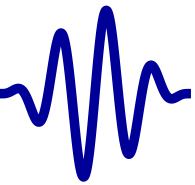
    `@$(PREFIX)_data`, `@$(PREFIX)_sel`
  - As well as bus return lines defined:

    `@$(PREFIX)_idata`, `@$(PREFIX)_ack`,

    `@$(PREFIX)_stall`

This is all in C++, and easily inheritable to support other bus types.

Interrupts



First, define the interrupt controller

- `@PREFIX`= the interrupt controllers name
- `@PIC.BUS`= names a group of interrupt wires
- `@PIC.MAX`= specifies the size of the wire group

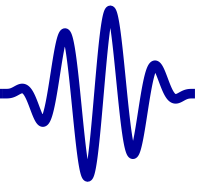
```
wire [@$(PIC.MAX)-1:0] @$(PIC.BUS);
```

Then, add interrupt wires to the bus

- `@INT.X.WIRE`= the name of X's interrupt wire
- `@INT.X.PIC`= the name of the PIC (`@PREFIX`) handling interrupt X (multiple PIC's are allowed)
- `@INT.X.ID`= an optional predefined interrupt ID

```
assign @$(PIC.BUS)[@$(INT.X.ID)] = @$(INT.X.WIRE);
```

Interrupts



Define an example controller:

@PREFIX=syspic

@PIC.BUS= sys_int_vector

@PIC.MAX= 15

Add interrupt wires

@INT.UARTRX.WIRE= uartrx_int

@INT.UARTRX.PIC= sysint

@INT.UARTRX.ID // let autofpga assign the ID

@INT.FLASH.WIRE= flash_interrupt

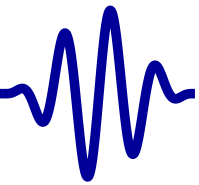
@INT.FLASH.PIC= sysint

@DEF.DEFN=

// Declare this interrupt to the ZipCPU library S/W

```
#define  INT_UARTRX  SYSINT(@$(INT.UARTRX.ID))
```

Clock Support



Clocks are easy to define. From within any component,

```
@CLOCK.NAME=clk
# @CLOCK.WIRE= // defaults to i_@$(CLOCK.NAME)
@CLOCK.FREQUENCY=100000000 // 100 MHz
```

Once defined within your design, placing:

```
@CLOCK.NAME=clk
```

copies the clock definition into your component.

Example:

```
@CLOCK.NAME=clk
@$BAUDCLOCKS=@$(CLOCK.FREQUENCY) / @$(BAUDRATE)
```

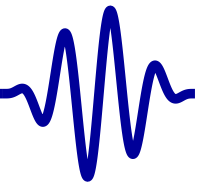
SIM Support

AutoFPGA builds a Verilator class that can be used to simulate multiple components, with multiple clocks.

```
@SIM.CLOCK=clk
@SIM.INCLUDE=
#include "uartsim.h"
@SIM.DEFNS=
UARTSIM *m_uart;
@SIM.INIT=
m_uart = new UARTSIM(0,@$(UARTSETUP));
@SIM.TICK= // Do the following C++ on each clock tick
m_core->i_uart_rx = m_uart->tick(m_core->o_uart_tx);
```

Remember: AutoFPGA is primarily a copy/paste utility

Inheritance



You can include files within a component

```
@INCLUDEFILE=parentclass.txt
```

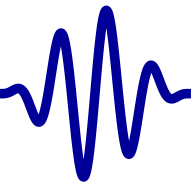
Keys from included files will be used anytime the key is not defined in the including file.

This means that adding a second UART, named `aux_uart`, on a different bus (`other` instead of `wb`), could be as simple as:

```
@PREFIX=aux_uart
@SLAVE.BUS=other
@INCLUDEFILE=uart.txt
```

To do this, values that can be overridden may need to include references to the components `@PREFIX` and `@SLAVE.BUS.NAME`.

Inherited

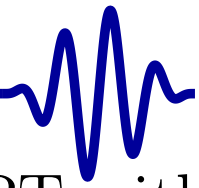


Inheritance requires replacing any unique names with parameters that may then be overwridden:

```
@MAIN.INSERT=
  wbuart #(.INITIAL_SETUP(@$(UARTSETUP))
    @$(PREFIX)i(@$(CLOCK.WIRE), 1'b0,
      // Bus inputs
      @$(BUS.NAME)_cyc, (@$(BUS.NAME)_stb)&&(@$(PREFIX)_sel), @$(BUS.NAME)_we,
          @$(BUS.NAME)_addr[1:0], @$(BUS.NAME)_data,
      // Bus outputs
      @$(PREFIX)_ack, @$(PREFIX)_stall, @$(PREFIX)_data,
      // Other wires
      i_@$(PREFIX)_rx, o_@$(PREFIX)_tx, @$(CTS), @$(RTS),
      // Interrupts
      @$(PREFIX)rx_int, @$(PREFIX)tx_int,
      @$(PREFIX)rxf_int, @$(PREFIX)txf_int);
```

Component specific names now use `@$(PREFIX)`, the clock is referenced by `@$(CLOCK.WIRE)`, and the name of the bus with `@$(BUS.NAME)`. All three may now be adjusted by inheritance.

Inheritance Ex



Suppose we wanted to create a new (or secondary) UART with neither RTS nor CTS wires.

```
@PREFIX=other_uart
# This core expects RTS and CTS to be active low
@CTS=1'b0
@RTS=1'b0
# Don't duplicate interrupts
@INT.INTLIST=
# Remove RTS and CTS from the portlist
@MAIN.PORTLIST=
  i_@$(PREFIX)_rx, o_@$(PREFIX)_tx
# and from the I/O declaration(s)
@MAIN.IODECL=
  input  wire i_@$(PREFIX)_rx;
  output wire o_@$(PREFIX)_tx;
@INCLUDEFILE=wbuart.txt
```

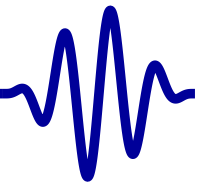
Current Status

The good: AutoFPGA is a GPLv3 technology demonstrator

- *I like it!* It is now a vital component of my new designs
- Adding a wire here or a wire there is *really* easy

  Knowing some basic keys, together with the Verilog and C++ languages is all that is necessary
- Ad-hoc components are easily added–and removed
- New designs can be quickly created from existing designs
- Removing a component is as easy as adjusting the command line, or the conditional synthesis defines
- Insulates your design from toolchain updates, since you can update AutoFPGA independently

Current Status



The bad: As of 201708, AutoFPGA is still immature

- It was been built rapidly to poorly defined requirements

  (The internal code still needs some care and feeding)

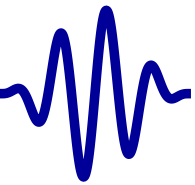
- Undergoing a lot of active development

  Please consider yourself invited to join in!

- The component file ICD is only mostly stable

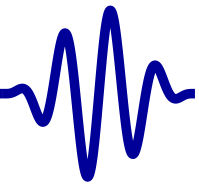
- While inheritance works, it can be harder to read an inheritable component file

Future Work



- Additional bus support (WB/B3, AXI4, etc.)
- GPIO aggregation
- LaTeX specification composition
- Linux Device Tree file generation
- GIT integration (perhaps via FuseSOC?)
- Components that do (and don't) specify their addresses

Example Designs



- Basic ZipCPU system

  `https://github.com/ZipCPU/zbasic`

- ICO Board

  `https://github.com/ZipCPU/icozip`

- Nexys Video project

  Still soliciting donations: the project will both read an HDMI stream into memory, as well as generate an HDMI stream from memory—subject to clock and memory bandwidth limitations

Gisselquist
Technology, LLC

In all labour there is profit . . .

Prov 14:23a