PipelineC

github.com/JulianKemmerer/PipelineC

Easier hardware description between RTL and HLS

HAFDAL '24 Workshop:

Hardware Acceleration of

Functional and Declarative Languages

~Context + Semantics

Write “C”

->

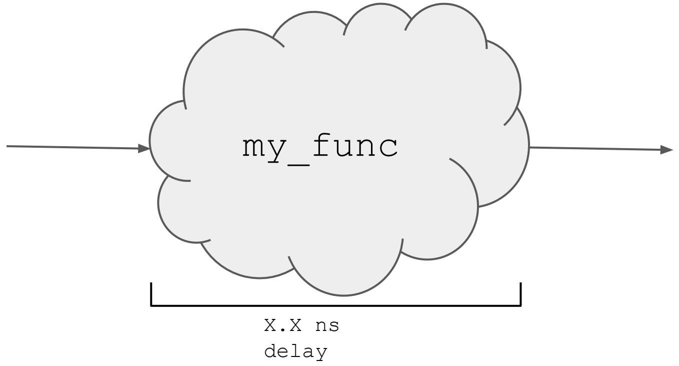
Hardware Description Language
(HDL=VHDL or Verilog)

Between register transfer level (RTL)
and full high level synthesis (HLS)

Pure Functions as Comb. Logic

HDLs know this: limited 'function' from VHDL/Verilog...
PipelineC combines functions+modules from HDL

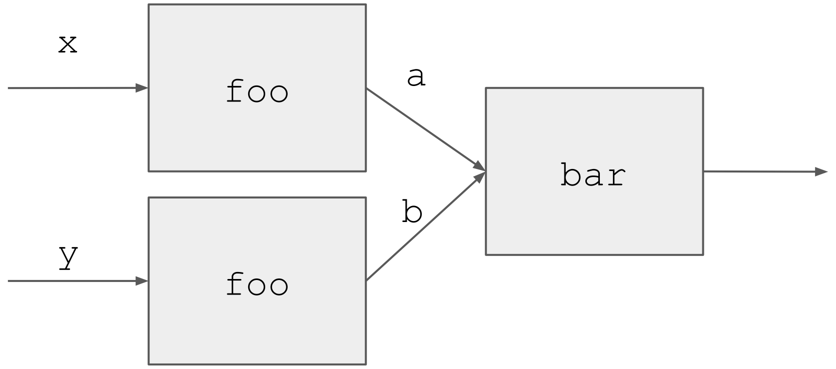
```
uint8_t my_func(uint8_t input_name)
{
  return ...
}
```



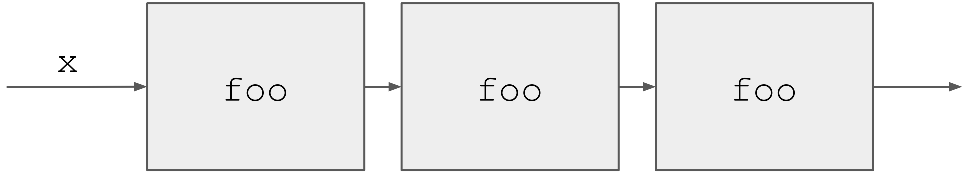
```
-- Generated PipelineC VHDL
entity my_func is
port(
  input_name :
     in unsigned(7 downto 0);
  return_output :
     out unsigned(7 downto 0)
);
end my_func;
```

Comb. Logic Dataflow Semantics

```
// Two parallel foo() instances
// one bar() instance
uint8_t my_func(uint8_t x, uint8_t y)
{
  uint8_t a = foo(x);
  uint8_t b = foo(y);
  return bar(a, b);
}
```



```
// Three iteration loop unrolled
uint8_t my_func(uint8_t x)
{
  for(i=0;i<3;i+=1)
  {
    x = foo(x);
  }
  return x;
}
```

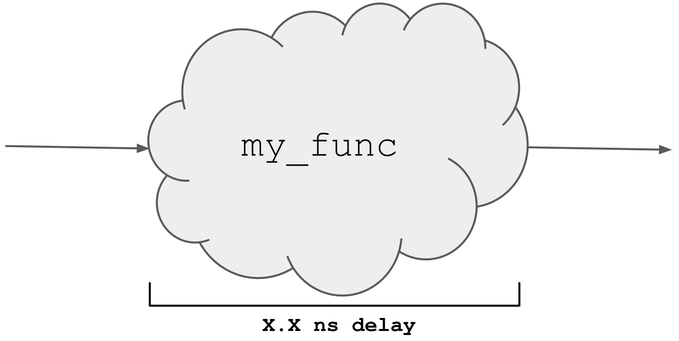


PipelineC Features:

Getting device specific timing feedback...

- Works with many tools
  - Xilinx Vivado
  - Intel Quartus
  - Lattice Diamond
  - GHDL+Yosys+nextpnr
  - Gowin EDA
  - Efinix Efinity
  - PyRTL ASIC Timing Models

```
uint8_t my_func(uint8_t input_name)
{
  return ...
}
```

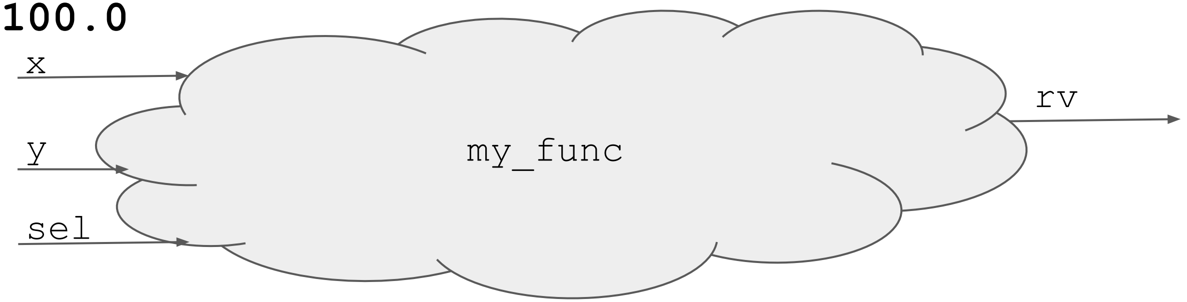


This device specific timing information allows autopipelining!

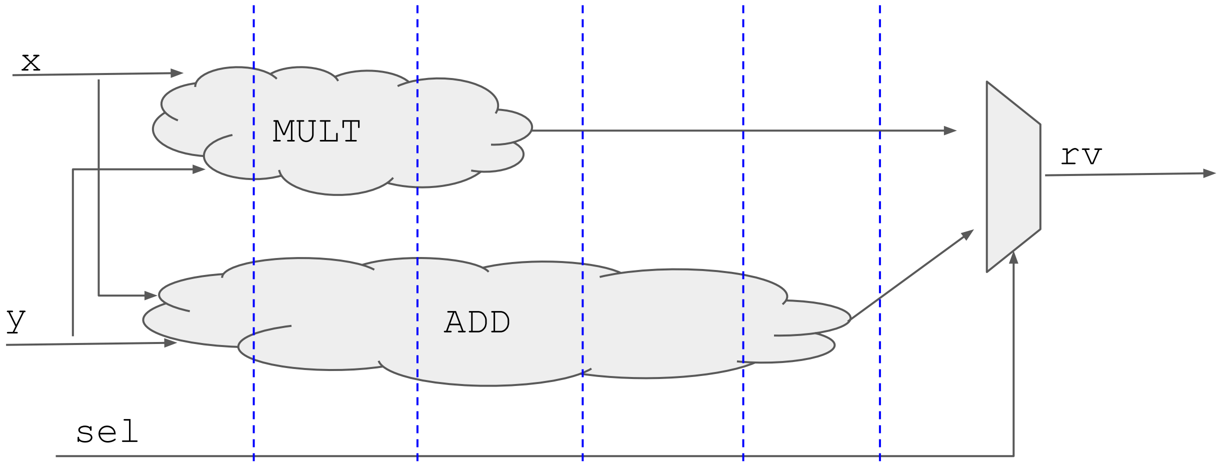
Comb. logic can be autopipelined

HLS-like, but PipelineC only does **II=1** for now...

```
#pragma PART "xc7a100tcsg324-1"
#pragma MAIN_MHZ my_func 100.0

float my_func(
  float x,
  float y,
  uint1_t sel
){
  float rv;
  if(sel){
   rv = x*y;
  }
  else{
   rv = x+y;
  }
  return rv;
}
```



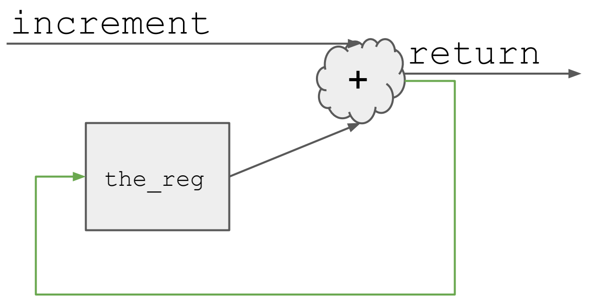
Pipeline stages depend on target device + fmax (not user specified staging)



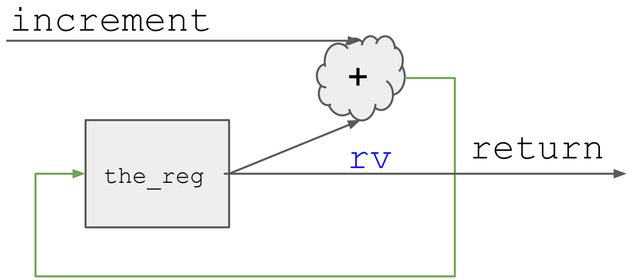
Comb. logic can be used with registers:

“RTL Style” **static** state registers, ~repeating impure functions, single clock domain

```
uint8_t my_counter(
  uint8_t increment
){
  static uint8_t the_reg;
  the_reg += increment;
  return the_reg;
}
```



```
uint8_t my_counter(
  uint8_t increment
){
  static uint8_t the_reg;
  uint8_t rv = the_reg;
  the_reg += increment;
  return rv;
}
```

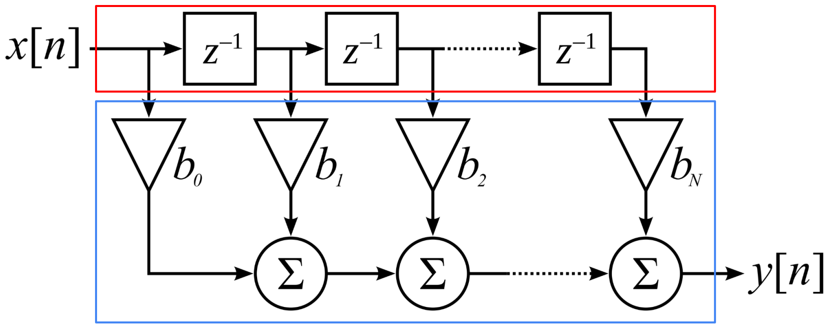


Clock by clock description: Will not autopipeline!

RTL Style

(one cycle at a time, not autopipelined)

FIR Filter

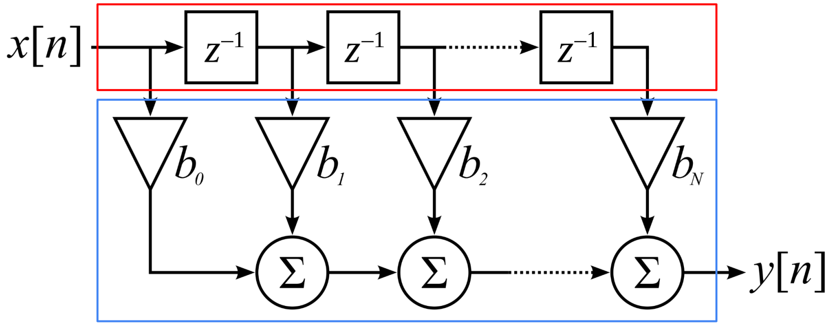


```
data_t fir(data_t x0)
{
      // Registers
      static data_t x[N];

      // Shift new x0 into array
      int32_t i;
      for(i=N-1; i>0; i-=1){
            x[i] = x[i-1];
      }
      x[0] = x0;

      // Multiply sample by coeff
      data_t b[N] = {...};
      data_t xb[N];
      for(i=0; i<N; i+=1){
            xb[i] = x[i]*b[i];
      }

      // Sum
      data_t sum = xb[0];
      for(i=1; i<N; i+=1){
            sum += xb[i];
      }
      return sum;
}
```

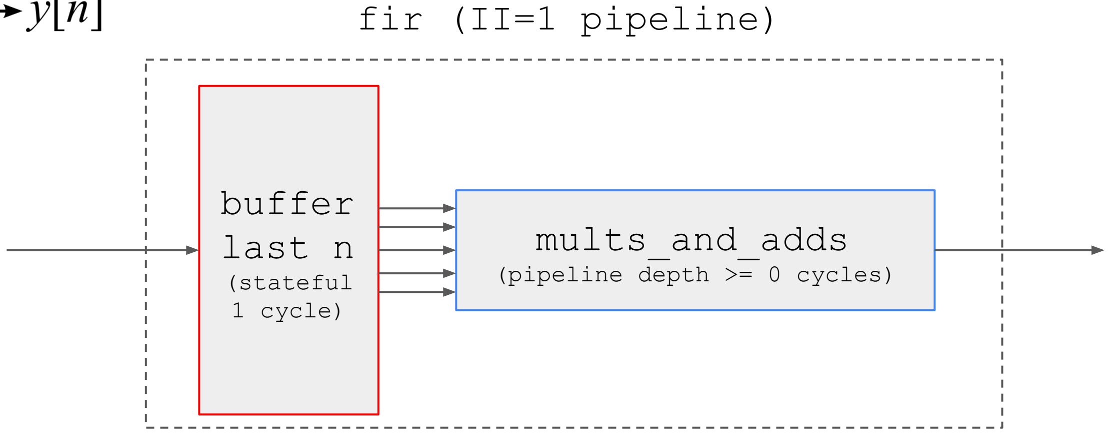
Pipeline Style

(dataflow, variable total latency)

FIR Filter



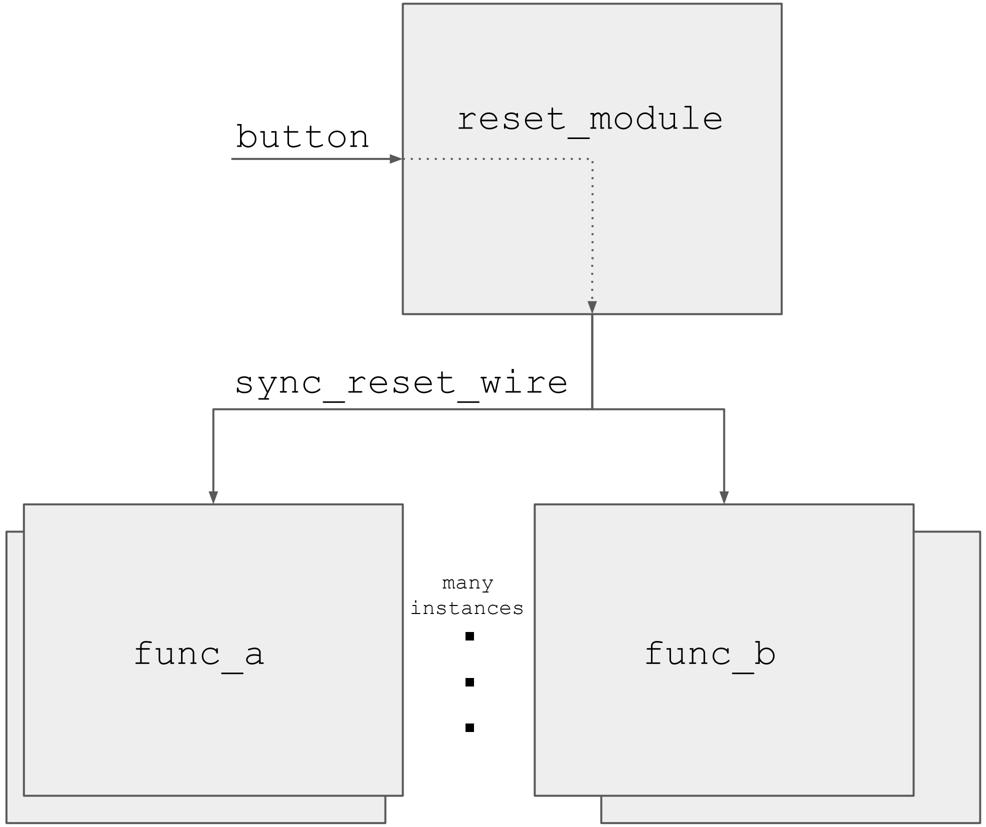
```
// Mix of stateful not-pipelined functions
// and stateless autopipeline-able functions
int16_t fir(int16_t x0)
{
  int16_t b[N] = {...};
  x = buffer_last_n(x0); // Impure
  y0 = mults_and_adds(b, x); // Pure
  return y0;
}
```



Global Variables:

Write-Once Read-Many Example

```
// Globally defined+visible
uint1_t sync_reset_wire;

#pragma MAIN_MHZ reset 100.0
void reset(uint1_t button)
{
  // Sync+debounce button...
  ... = button;
  sync_reset_wire = ...;
}


void func_a()
{
  if(sync_reset_wire)...
}
void func_b()
{
  ... = sync_reset_wire;
}
```



FPGA Examples:

What can you do with?

Comb. logic,

registers,

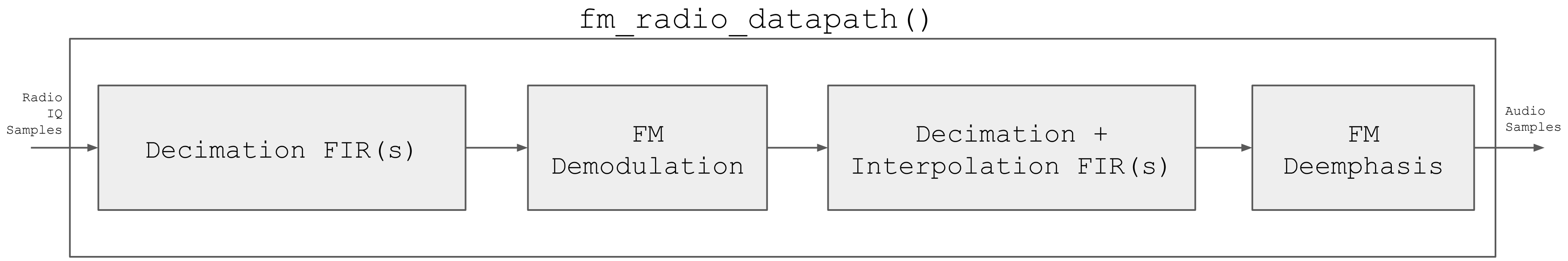
pipelines,

and wires?

Practically everything!

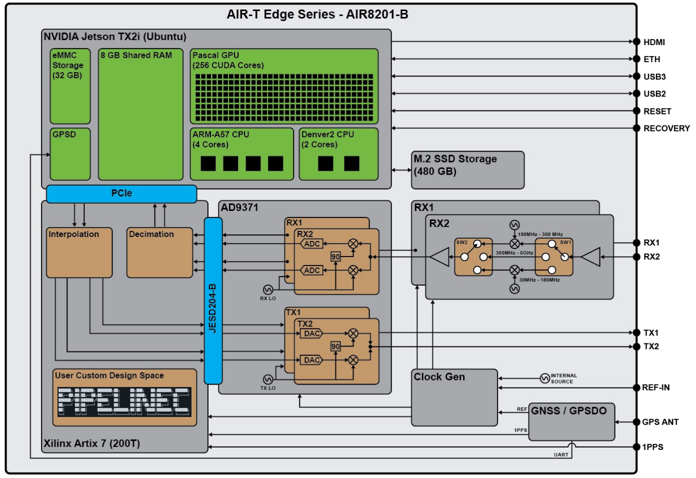
DSP Demo: First announcement!

FM Radio on FPGA SDR

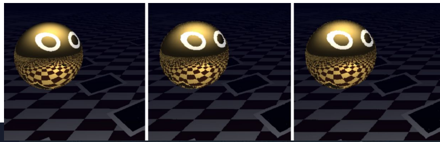


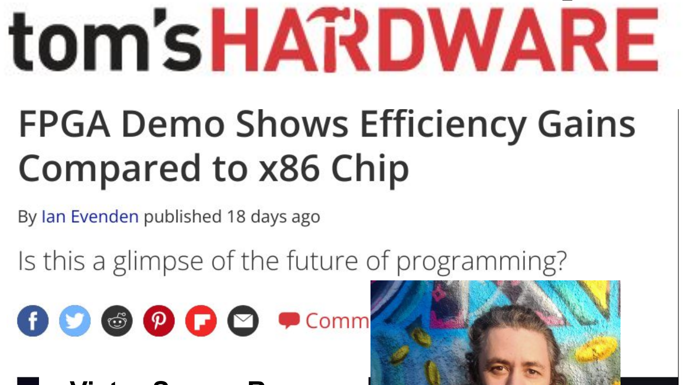
- Single top level MAIN function for data path
  - Single clock domain
- Auto-pipelined to 125MHz timing goal
  - ~32 stage pipeline on Xilinx Artix 7

- Deepwave Digital AIR-T 8201
  - AirStack Sandbox FPGA Dev Kit
  - 125MSPS RF front end
- SoapySDR Python API



Sphery vs. Shapes





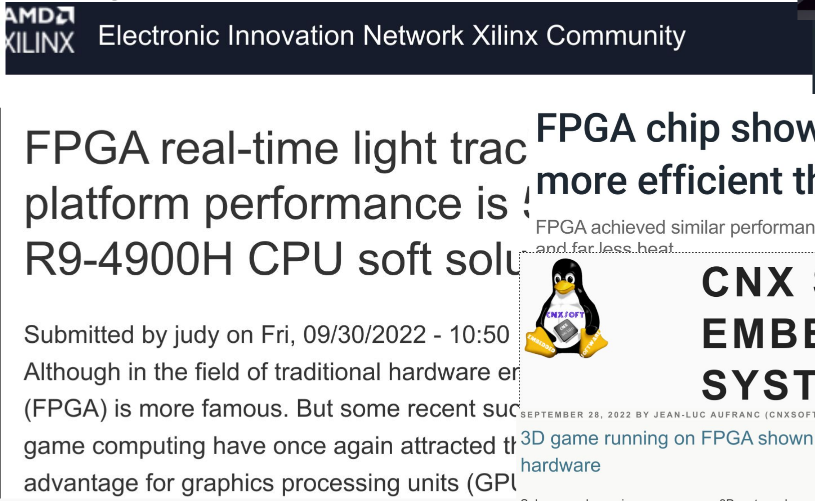
tom's HARDWARE

FPGA Demo Shows Efficiency Gains Compared to x86 Chip

By Ian Evenden published 18 days ago

Is this a glimpse of the future of programming?

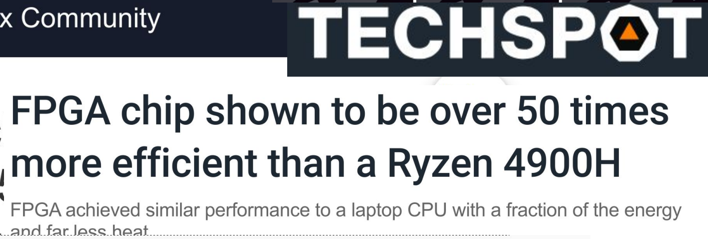
Comm



AMD XILINX Electronic Innovation Network Xilinx Community

FPGA real-time light trac
platform performance is
R9-4900H CPU soft solu

Submitted by judy on Fri, 09/30/2022 - 10:50
Although in the field of traditional hardware er
(FPGA) is more famous. But some recent suc
game computing have once again attracted tl
advantage for graphics processing units (GPU



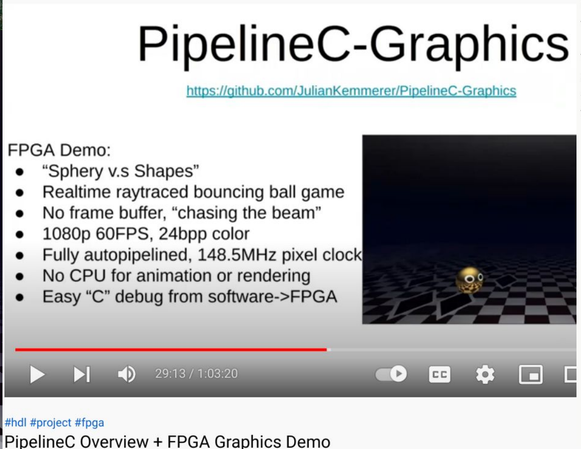
TECHSPOT

FPGA chip shown to be over 50 times more efficient than a Ryzen 4900H

FPGA achieved similar performance to a laptop CPU with a fraction of the energy and far less heat

Victor Suarez Rovere
@suarezvictor

YosysHQ Blog



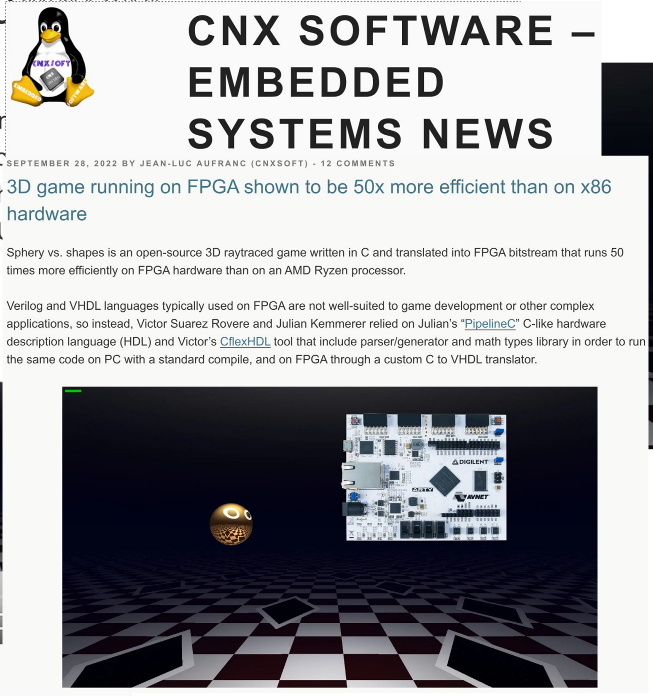
PipelineC-Graphics

https://github.com/JulianKemmerer/PipelineC-Graphics

FPGA Demo:
- "Sphery v.s Shapes"
- Realtime raytraced bouncing ball game
- No frame buffer, "chasing the beam"
- 1080p 60FPS, 24bpp color
- Fully autopipelined, 148.5MHz pixel clock
- No CPU for animation or rendering
- Easy "C" debug from software->FPGA

29:13 / 1:03:20

#hdl #project #fpga
PipelineC Overview + FPGA Graphics Demo



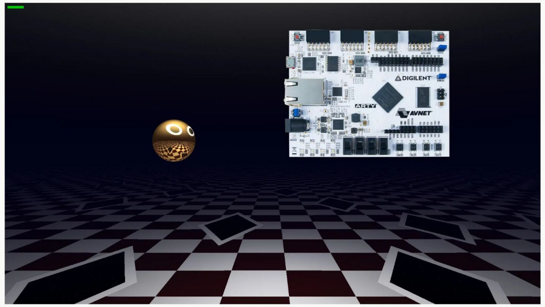
CNX SOFTWARE – EMBEDDED SYSTEMS NEWS

SEPTEMBER 28, 2022 BY JEAN-LUC AUFRANC (CNXSOFT) - 12 COMMENTS

3D game running on FPGA shown to be 50x more efficient than on x86 hardware

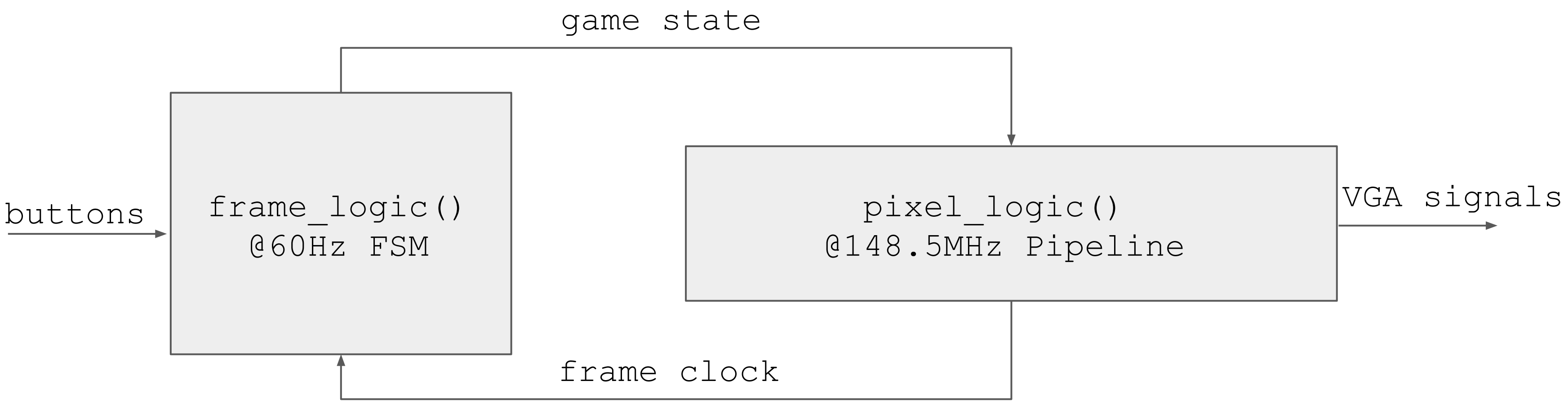
Sphery vs. shapes is an open-source 3D raytraced game written in C and translated into FPGA bitstream that runs 50 times more efficiently on FPGA hardware than on an AMD Ryzen processor.

Verilog and VHDL languages typically used on FPGA are not well-suited to game development or other complex applications, so instead, Victor Suarez Rovere and Julian Kemmerer relied on Julian's "PipelineC" C-like hardware description language (HDL) and Victor's CflexHDL tool that include parser/generator and math types library in order to run the same code on PC with a standard compile, and on FPGA through a custom C to VHDL translator.



Ray Traced Game: Top Level Design

- Two clock domains, ~two **`#pragma MAIN`** functions, two important global wires



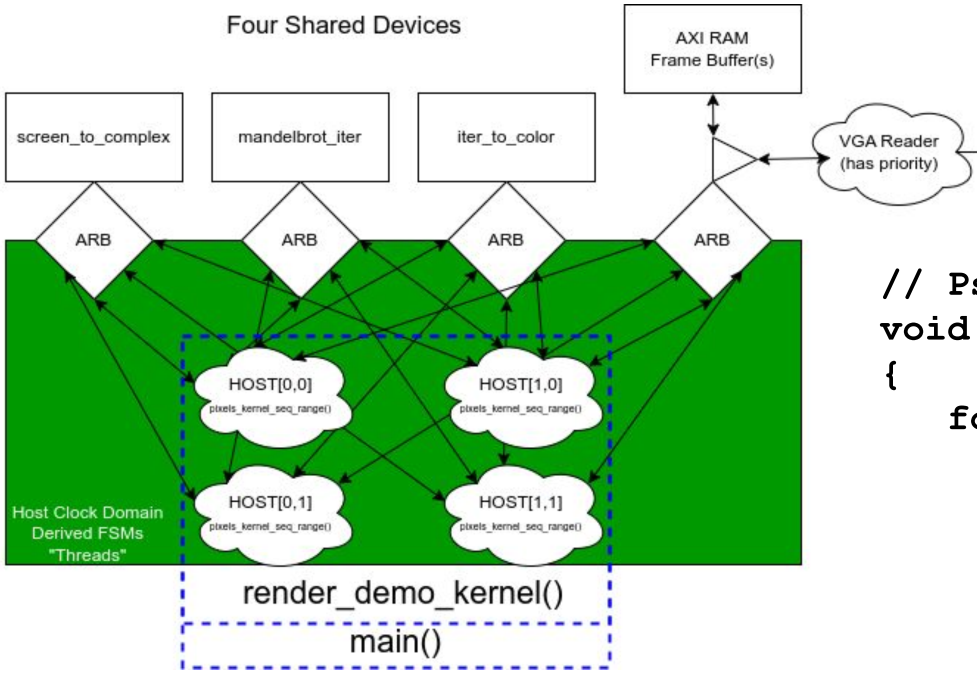
- Things occurring at 60Hz
- Per frame animation, object pos, etc
- **Very slow state machine**

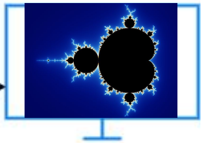
- Things occurring at pixel clock, 148.5MHz
- Generating VGA timing: h/vsync, x,y positions to render
  - ‘Chasing the beam’
- Generating frame clock, 60Hz
- **Hundreds of stages rendering pipeline from C function**

```
pixel_t render_pixel(uint16_t x, uint16_t y)
```

Shared resources bus demo:

Multiple FSMs sharing multiple pipelines





Drawing the Mandelbrot set

```
// Pseudo code using derived FSMs
void render_thread(xpos, xsize, ypos, ysize)
{
    for(x=xpos...){ // Not unrolled,
      for(y=ypos...){ // derives FSM
        complex = screen_to_complex(x,y);
        iters = mandelbrot(complex);
        color = iter_to_color(iters)
        frame_buffer_write(x, y, color);
      }
    }
}
```

RISC-V cores next?

Future Work

- Improvements to autopipelining
  - Report+optimize for area/resources
  - Dealing with feedback / stall signals
- Improved derived finite state machines
  - Better code generation
  - RISC-V integration
- Template types+functions:
  - How to parameterize functions?
    - Compile time computation
  - Currently ugly preprocessor hacks
- ‘Compile entire PipelineC designs with software compilers’
  - Built in ultra-fast simulations
- Modern hardware compiler frameworks / intermediates + tools

Thanks folks! Questions? 🤓

https://github.com/JulianKemmerer/PipelineC/wiki

https://github.com/JulianKemmerer/PipelineC/wiki/Example:-FM-Radio-Demodulation

https://deepwavedigital.com/ - SDR Platform <3

https://github.com/JulianKemmerer/PipelineC-Graphics - Sphery vs. Shapes Ray Tracer

https://github.com/JulianKemmerer/PipelineC/wiki/Shared-Resource-Bus - Mandelbrot

Mastodon: pipelinec@fosstodon.org

Talk on the PipelineC Discord: https://discord.gg/Aupm3DDrK2