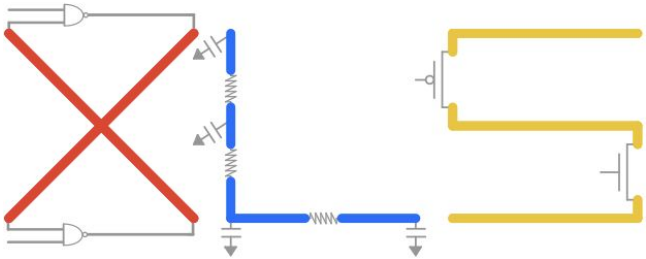
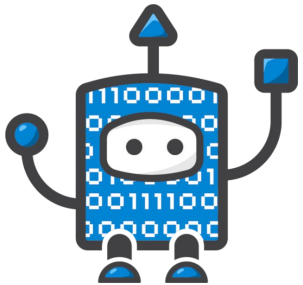
Porting Software to Hardware

with XLS/DSLX on the FoMU





@proppy and Rob Springer

Meet the FoMu

FPGA: Lattice ICE40UP5K

LUTs: 5280

Speed: 48 MHz external oscillator

RAM: 128 kB RAM

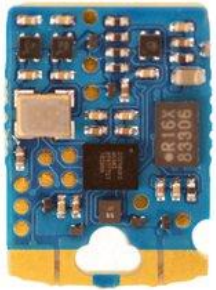
Storage: 2 MB SPI flash

Connectivity: USB 2.0

Peripherals: 4x touchpads, 1x RGB LED

OSS Toolchain: symbiflow.github.io

Getting Started: workshop.fomu.im



Blink a LED (with Software)

MicroPython

C

CSR

RISC-V

FPGA

SB_RGBA_DRV

RGB1PWM

```
import fomu
import time

LEDDPWRR = 0b0001
rgb = fomu.rgb()

while True:
    rgb.write_raw(LEDDPWRR, 255)
    time.sleep_ms(500)
    rgb.write_raw(LEDDPWRR, 0)
    time.sleep_ms(500)
```

Blink a LED (with Hardware)

Verilog

FPGA

SB_RGBA_DRV

RGB1PWM

```
module top (
    input  clk,
    output rgb0,
    output rgb1,
    output rgb2,
    // ...
);
   reg [24:0] counter = 0;
   always @(posedge clk) begin
      counter <= counter + 1;
   end

   SB_RGBA_DRV #(
      // ...
    ) RGBA_DRIVER (
      // ...
      .`GREENPWM(0),
       // blink at 2Hz
      .`REDPWM(counter[24]),
      .`BLUEPWM(0),
      .RGB0(rgb0),
      .RGB1(rgb1),
      .RGB2(rgb2)
    );
endmodule
```

Cycle a LED (with Software)

C

`fast_hsv2rgb_32bit`

CSR

RISC-V

FPGA

SB_LEDDA_IP

SB_RGBA_DRV

RGB0PWM RGB1PWM RGB2PWM

```c
#define HSV_MONOCHROMATIC_TEST(s, v, r, g, b) \
  do {                                        \
    if (!(s)) {                               \
      *(r) = *(g) = *(b) = (v);               \
      return;                                 \
    }                                         \
  } while (0)

#define HSV_SEXTANT_TEST(sextant) \
  do {                            \
    if ((sextant) > 5) {          \
      (sextant) = 5;              \
    }                             \
  } while (0)

#define HSV_SWAPPTR(a, b) \
  do {                    \
    uint8_t *tmp = (a);   \
    (a) = (b);            \
    (b) = tmp;            \
  } while (0)

#define HSV_POINTER_SWAP(sextant, r, g, b) \
  do {                                     \
    if ((sextant)&2) {                     \
      HSV_SWAPPTR((r), (b));               \
    }                                      \
    if ((sextant)&4) {                     \
      HSV_SWAPPTR((g), (b));               \
    }                                      \
    if (!((sextant)&6)) {                  \
      if (!((sextant)&1)) {                \
        HSV_SWAPPTR((r), (g));             \
      }                                    \
    } else {                               \
      if ((sextant)&1) {                   \
        HSV_SWAPPTR((r), (g));             \
      }                                    \
    }                                      \
  } while (0)

void fast_hsv2rgb_32bit(uint16_t h, uint8_t s, uint8_t v, uint8_t *r, uint8_t *g , uint8_t *b) {
  uint8_t sextant = h >> 8;
  HSV_SEXTANT_TEST(sextant);
  HSV_POINTER_SWAP(sextant, r, g, b);
  *g = v;
  uint16_t ww;
  ww = v * (255 - s);
  ww += 1;
  ww += ww >> 8;
  *b = ww >> 8;
  uint8_t h_fraction = h & 0xff;
  uint32_t d;
  if(!(sextant & 1)) {
    d = v * (uint32_t)((255 << 8) - (uint16_t)(s * (256 - h_fraction)));
    d += d >> 8;
    d += v;
    *r = d >> 16;
  } else {
    d = v * (uint32_t)((255 << 8) - (uint16_t)(s * h_fraction));
    d += d >> 8;
    d += v;
    *r = d >> 16;
  }
}
```

Cycle a LED (with Hardware)

Verilog

???

FPGA

SB_LEDDA_IP

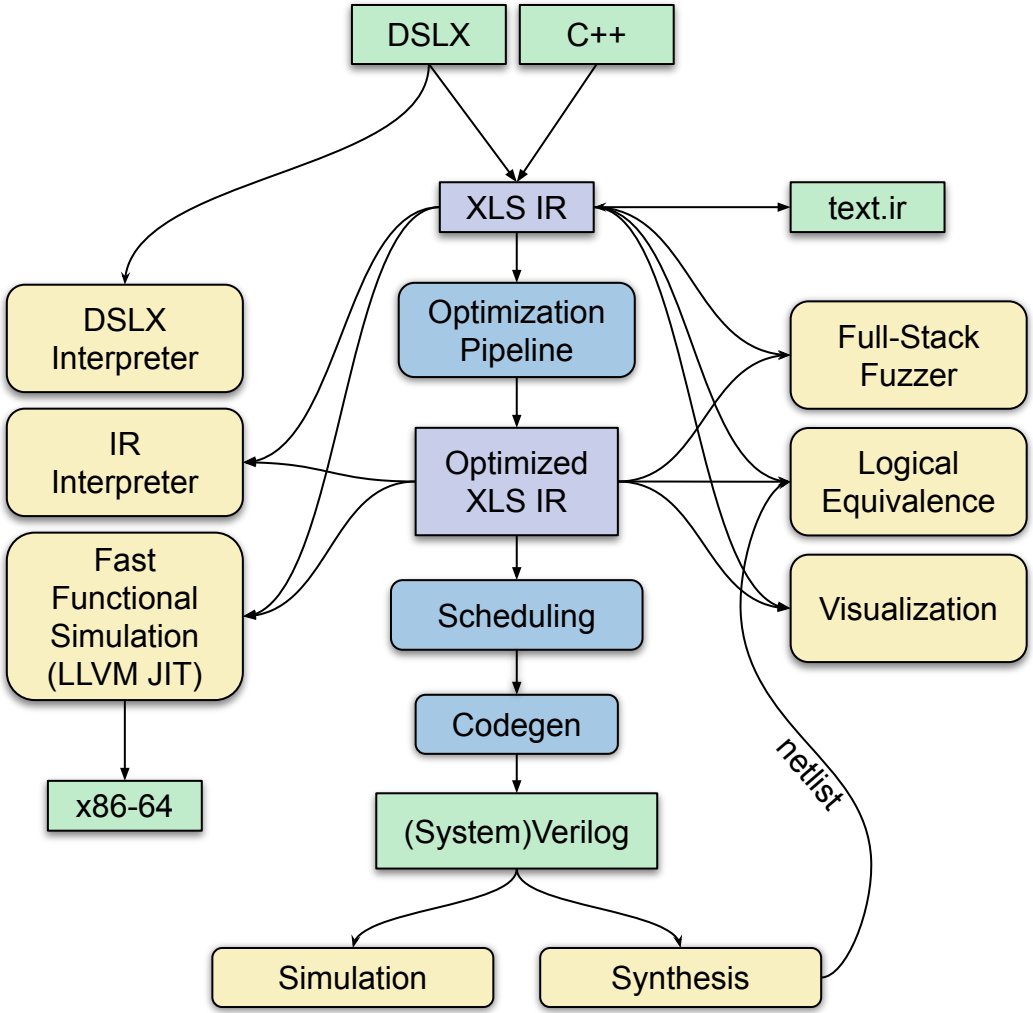
SB_RGBA_DRV

RGB0PWM RGB1PWM RGB2PWM

```
module top (
    input  clk,
    output rgb0,
    output rgb1,
    output rgb2,
    // ...
);

   // INSERT
   // QUITE
   // A
   // LOT
   // OF
   // VERILOG
   // HERE

   SB_RGBA_DRV #(
      // ...
    ) RGBA_DRIVER (
      // ...
      .`GREENPWM(g),
      .`REDPWM(r),
      .`BLUEPWM(b),
      .RGB0(rgb0),
      .RGB1(rgb1),
      .RGB2(rgb2)
    );
endmodule
```

Meet XLS!

- A *high-level synthesis (HLS) toolchain*
- *Turns SWEs into HWEs*
- Compiles designs to Verilog or *native-speed* libraries
- Provides DSL, interpreters, logical tools, auto-pipelining, ...
- google.github.io/xls



DSLX

fast_hsv2rgb.c → hsv2rgb.x

- dataflow DSL
- mimics Rust
- immutable, expression-based
- fully analyzable call graph
- arbitrary bit widths
- fixed size types
- parametric <T> functions

```
pub fn slope(h_fraction: u16, s: u8, v:u8) -> u8 {
  let sh_fraction:u32 = ((s as u16) * h_fraction) as u32);
  let u:u32 = (v as u32) * ((u32:255 << 8) - sh_fraction;
  let u:u32 = u + (u >> 8);
  let u:u32 = u + (v as u32);
  u[16:24]
}

pub fn hsv2rgb(h: u16, s: u8, v: u8) -> (u8, u8, u8) {
  let sextant:u8 = h[8:16];
  let sextant:u8 = if sextant > u8:5 { u8:5 } else { sextant };

  let ww:u16 = (v as u16) * ((u8:255 - s) as u16);
  let ww:u16 = ww + u16:1;
  let ww:u16 = ww + ww >> 8;
  let c:u8 = ww[8:16];

  let h_fraction:u16 = h[0:8] as u16;
  let nh_fraction:u16 = u16:256 - h_fraction;
  let u:u8 = slope(nh_fraction, s, v);
  let d:u8 = slope(h_fraction, s, v);

  if s == u8:0 { (v, v, v) }
  else {
    match sextant {
      u8:0 => (v, u, c),
      u8:1 => (d, v, c),
      u8:2 => (c, v, u),
      u8:3 => (c, d, v),
      u8:4 => (u, c, v),
      u8:5 => (v, c, d),
      _ => fail!((u8:0, u8:0, u8:0)),
    }
  }
}
```

out pointers w/ early return → tuple tail expression

C	DSLX

`void fast_hsv2rgb_32bit(`
`uint16_t h, uint8_t s, uint8_t v,`
`uint8_t *r, uint8_t *g , uint8_t *b`
`) {`
`  if (...) {`
`  return;`
`  }`
 	
`  // ...`
 	
`  return;`
`}`
	`pub fn hsv2rgb(h: u16, s: u8, v: u8)`
	`-> (u8, u8, u8) {`
	
	`  // ...`
	
	`  (r, g, b)`
	`}`

bit mask and shift → bit slice expressions

C	DSLX
`uint8_t sextant = h >> 8;`	`let sextant:u8 = h[8:16];`
`uint8_t h_fract = h & 0xff;`	`let h_fract:u8 = h[0:8];`
`uint8_t sextant = h >> 8;`	`let sextant:u8 = h[8:+8];`
`uint8_t sextant = h >> 8;`	`let sextant:u8 = h[-8:];`

binary literal and concat

C	DSLX
`0xff`	`u8:0b11111111`
`0xff`	`u8:0b1111_1111`
`0xff`	`u4:0b1111 ++ u4:0b1111`

mut → lexically scoped let expression

C	DSLX

`w = v * (255 - s);`
`w += 1;`
`w += w >> 8;`
	`let w:u16 = v * (u16:255 - s);`
	`let w:u16 = w + u16:1;`
	`let w:u16 = w + w[-8:];`

`w = v * (255 - s);`
`w += 1;`
`w += w >> 8;`
	`let w:u16 = v * (u16:255 - s);`
	`let w':u16 = w + u16:1;`
	`let w'':u16 = w' + w'[-8:];`

`uint8_t sextant = h >> 8;`
`if (sextant > 5) {`
`  sextant = 5;`
`}`
	`let sextant:u8 =`
	`  if h[-8:] > u8:5 { u8:5 }`
	`  else { h[-8:] };`

branch→ match/if exprs

C	DSLX

`if (!(s)) {`
`  *(r) = *(g) = *(b) = (v);`
`  return;`
`}`
 	
`// ...`
 	
`if ((sextant)&2) {`
`  HSV_SWAPPTR((r), (b));`
`}`
`if ((sextant)&4) {`
`  HSV_SWAPPTR((g), (b));`
`}`
`if (!((sextant)&6)) {`
`  if (!((sextant)&1)) {`
`  HSV_SWAPPTR((r), (g));`
`  }`
`} else {`
`  if ((sextant)&1) {`
`  HSV_SWAPPTR((r), (g));`
`  }`
`}`
	`if s == u8:0 { (v, v, v) }`
	` else {`
	`  match sextant {`
	`  u8:0 => (v, u, c),`
	`  u8:1 => (d, v, c),`
	`  u8:2 => (c, v, u),`
	`  u8:3 => (c, d, v),`
	`  u8:4 => (u, c, v),`
	`  u8:5 => (v, c, d),`
	`  _ => fail!((u8:0, u8:0, u8:0)),`
	`  }`
	` }`

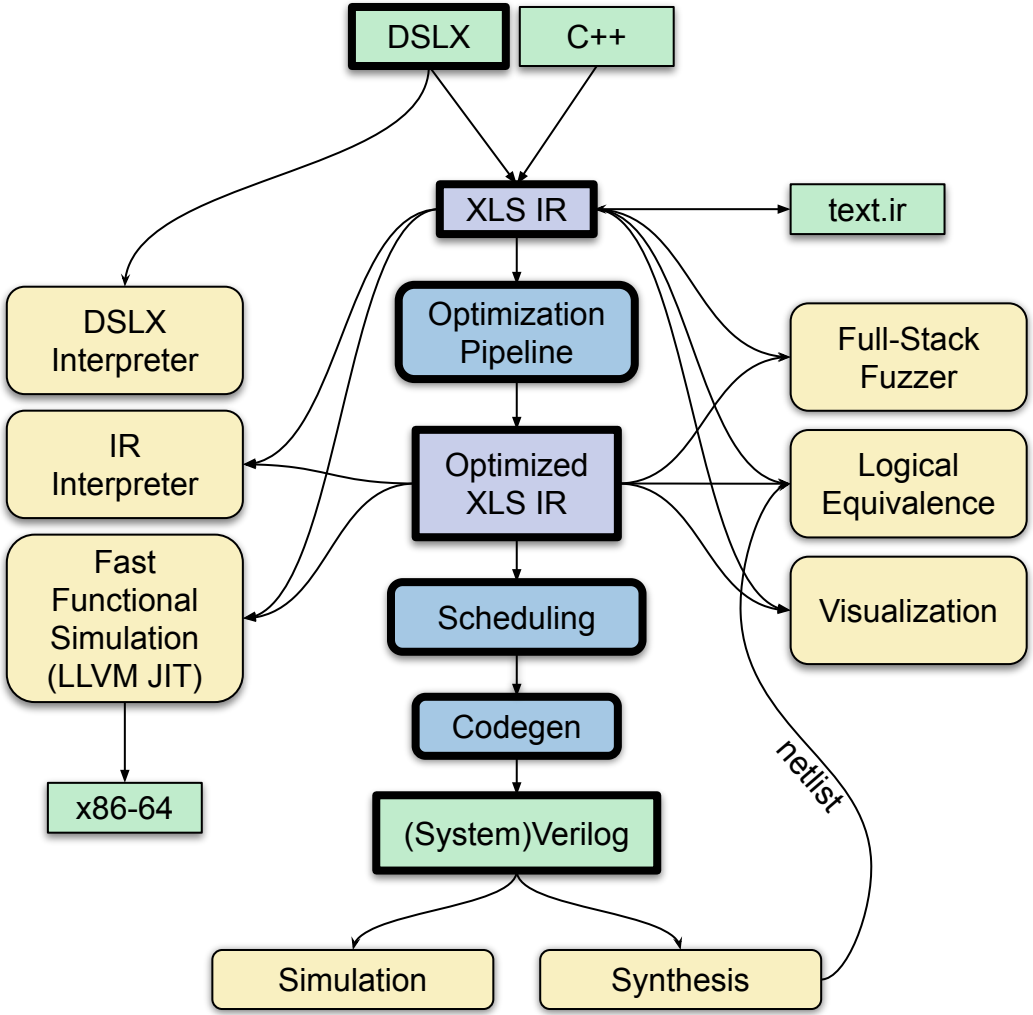
Testing

- `#![test]` for unittests
- `#![quickcheck]` for property based testing
- embedded in DSLX
    - test w/ DSLX interpreter
    - test w/ XLS IR + JIT
- C++ test using XLS API
    - tested against C implementation
    - tested using verilog simulator
- formal verification using Z3

```
#![test]
fn hsv2rgb_test() {
  let _ = assert_eq(
    hsv2rgb(u16:0, u8:0, u8:0),
    (u8:0, u8:0, u8:0)
  );
  let _ = assert_eq(
    hsv2rgb(u16:0, u8:0, u8:255),
    (u8:255, u8:255, u8:255)
  );
  let _ = assert_eq(
    hsv2rgb(u16:300, u8:0, u8:127),
    (u8:127, u8:127, u8:127)
  );
  let _ = assert_eq(
    hsv2rgb(u16:0, u8:255, u8:255),
    (u8:255, u8:0, u8:0)
  );
  // ...
  _
}
```

XLS toolchain

1. User writes DSLX
2. DSLX is compiled into XLS IR
3. IR is optimized to simplify logic, remove unnecessary ops, etc.
4. IR is“scheduled”: placed in pipeline stages
5. IR is lowered into RTL

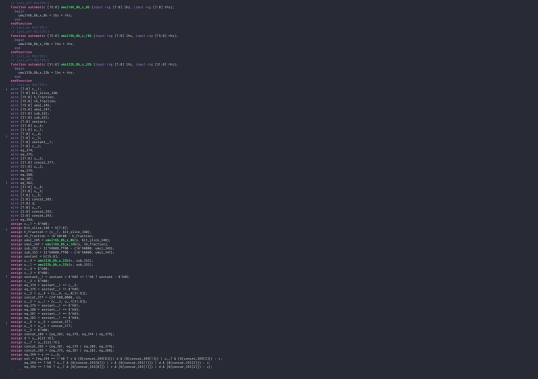


hsv2rgb.x → hsv2rgb_opt.v workflow

```
# Run tests
$ ./dslx/interpreter_main hsv2rgb.x

# Convert to IR
$ ./dslx/ir_converter_main hsv2rgb.x > hsv2rgb.ir

# Optimize IR
$ ./tools/opt_main hsv2rgb.ir > hsv2rgb_opt.ir

# Generate Verilog
./tools/codegen_main --generator=combinational \
                     hsv2rgb_opt.ir > hsv2rgb.v
```

putting it all together

```
// hsv2rgb_opt.v
module __hsv2rgb__hsv2rgb(
  input wire [15:0] h,
  input wire [7:0] s,
  input wire [7:0] v,
  output wire [23:0] out
);
```



```
endmodule
```

```
// top.v
`include "hsv2rgb.v"

module top (
  input  clki,
  output rgb0,
  output rgb1,
  output rgb2,
  /// ...
)

  /// ...
  __hsv2rgb__hsv2rgb hsv2rgb_1(
    counter[15:0], s, v, rgb
  );
  wire [7:0] r = rgb[23:16];
  wire [7:0] g = rgb[15:8];
  wire [7:0] b = rgb[7:0];

endmodule // top
```

github.com/google/xls
/tree/main/third_party/xls_colors

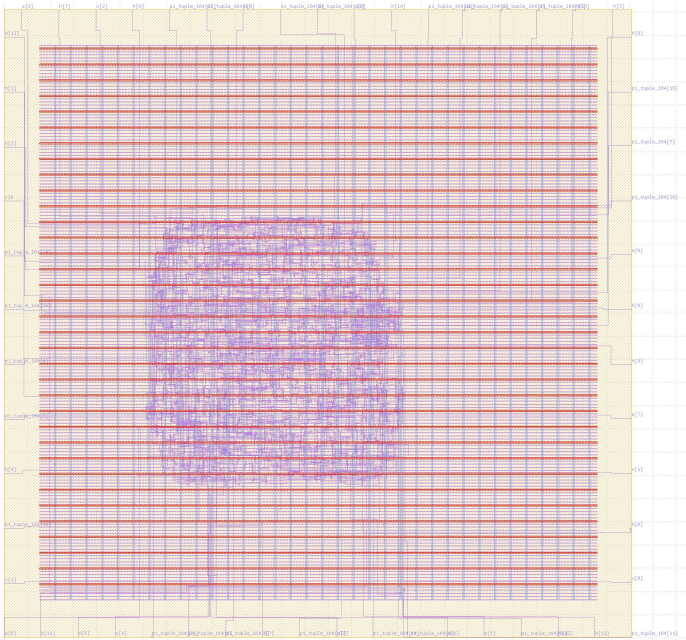
github.com/im-tomu/fomu-workshop
/tree/master/hdl/verilog/blink

Integrations

- **`xls/public/`**: public C++ API w/ Python Bindings
  verilog flow wip: github.com/google/xls/pull/484
- **`xls/build_rules/`**: bazel rules
  `xls_dslx_module_library xls_dslx_test xls_dslx_ir xls_ir_opt_ir xls_ir_verilog`
- **`third_party/`**: OSS ports
  `xls_machsuite (fft) xls_berkeley_softfloat xls_go_math, xls_colors`
- **`xls/examples/`**: DSLX examples
  `idct_chen.x sobel_filter.x fir_filter.x sha256.x crc32.x cubic_bezier.x riscv_simple.x hack_cpu.x`

github.com/hdl/bazel_rules_hdl

- combine HDL output with synthesis tool
- compose well with XLS rules
- target ASIC workflow w/ theopenroadproject.org
- manufacture test chips with skywater-pdk.rtfd.io



One more thing

- iCEBreaker
- PMOD LED Panel Driver
- 64x64 P2 LED Panel
- 5V 10A Power Supply