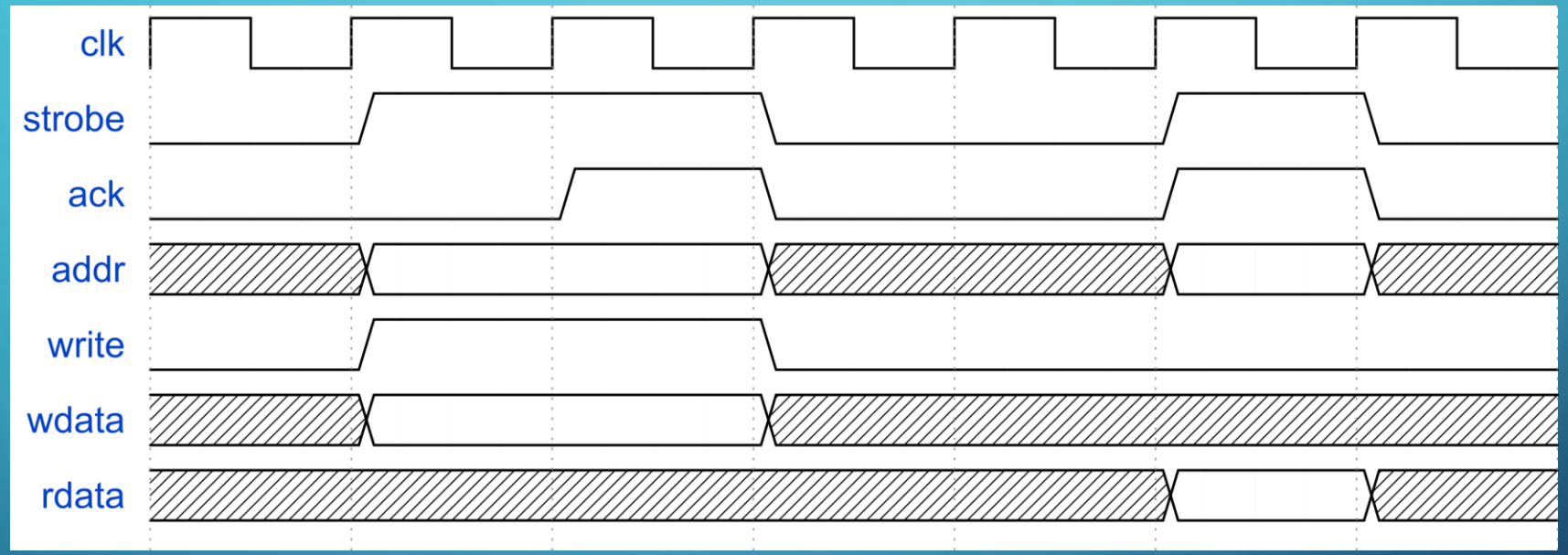
GETTING STARTED WITH THE EMP FRAMEWORK – PART 5

ANDY ROSE, IMPERIAL COLLEGE LONDON

IPBUS

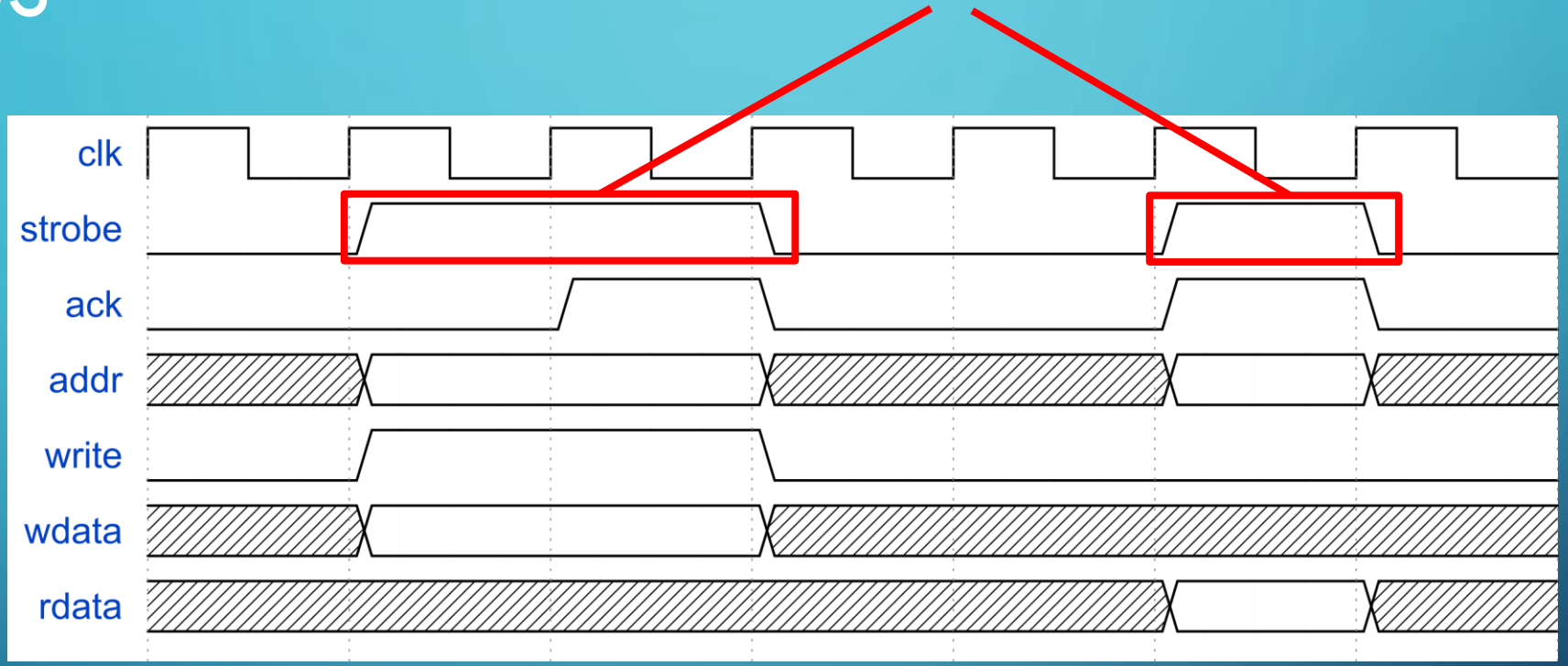
- Let’s configure something!

IPBUS



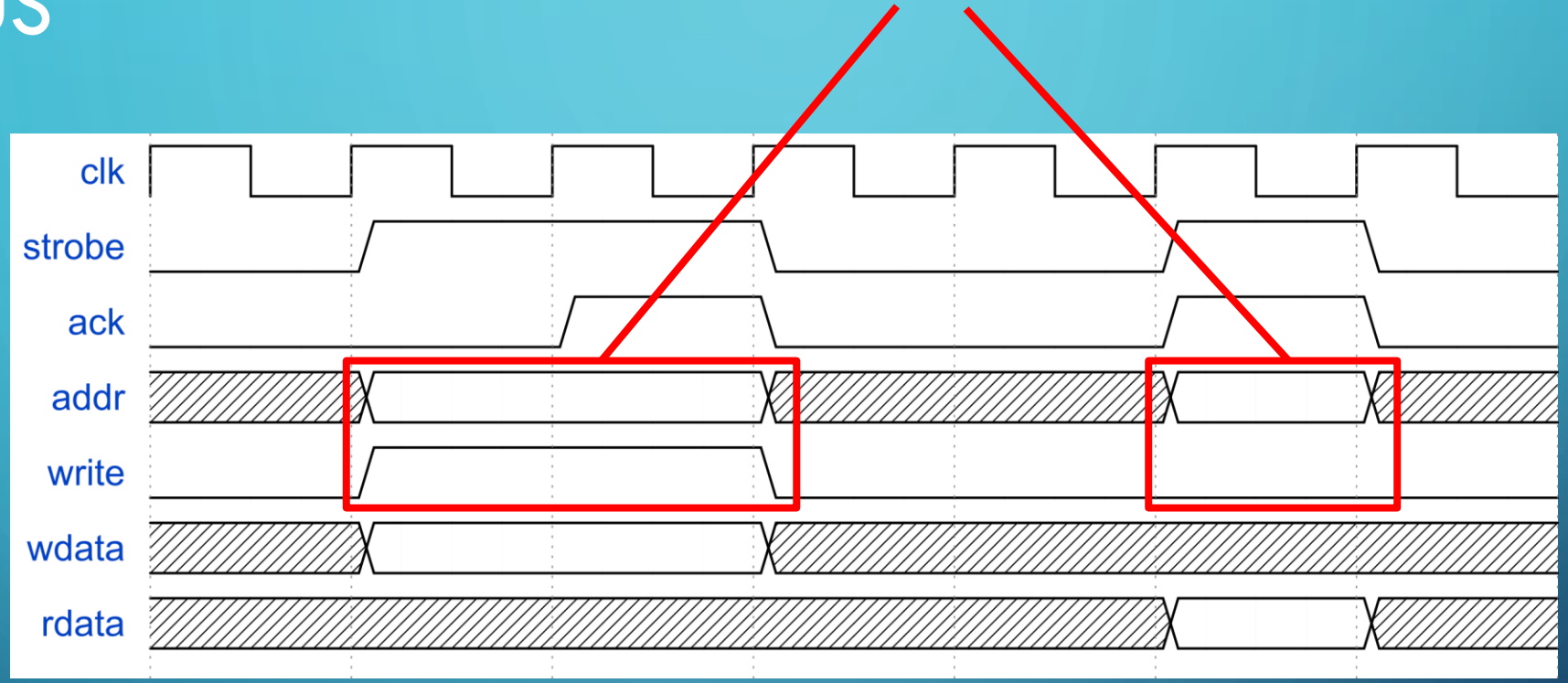
IPBUS

- Active transactions are marked by a high strobe from the master
- There does not need to be a low between active transactions



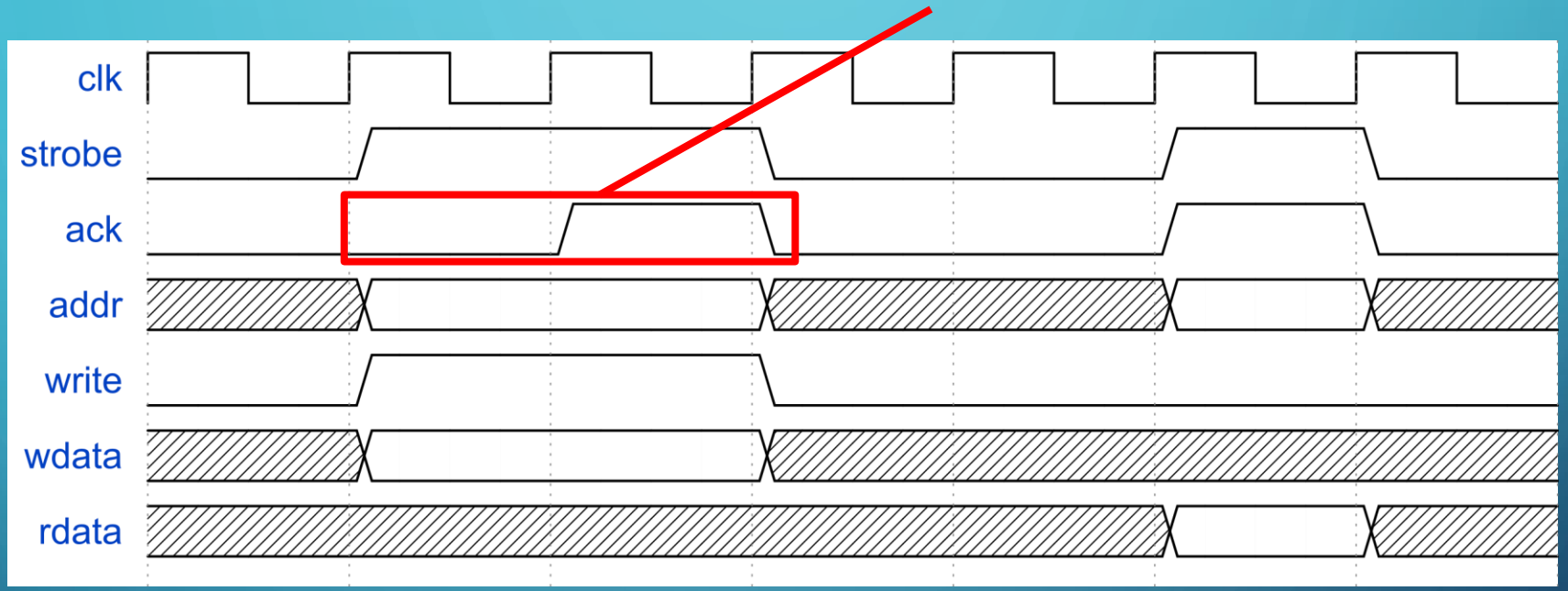
IPBUS

- The master includes an address to identify which endpoint it is talking to
- The master identifies read or write transactions with the “write” flag



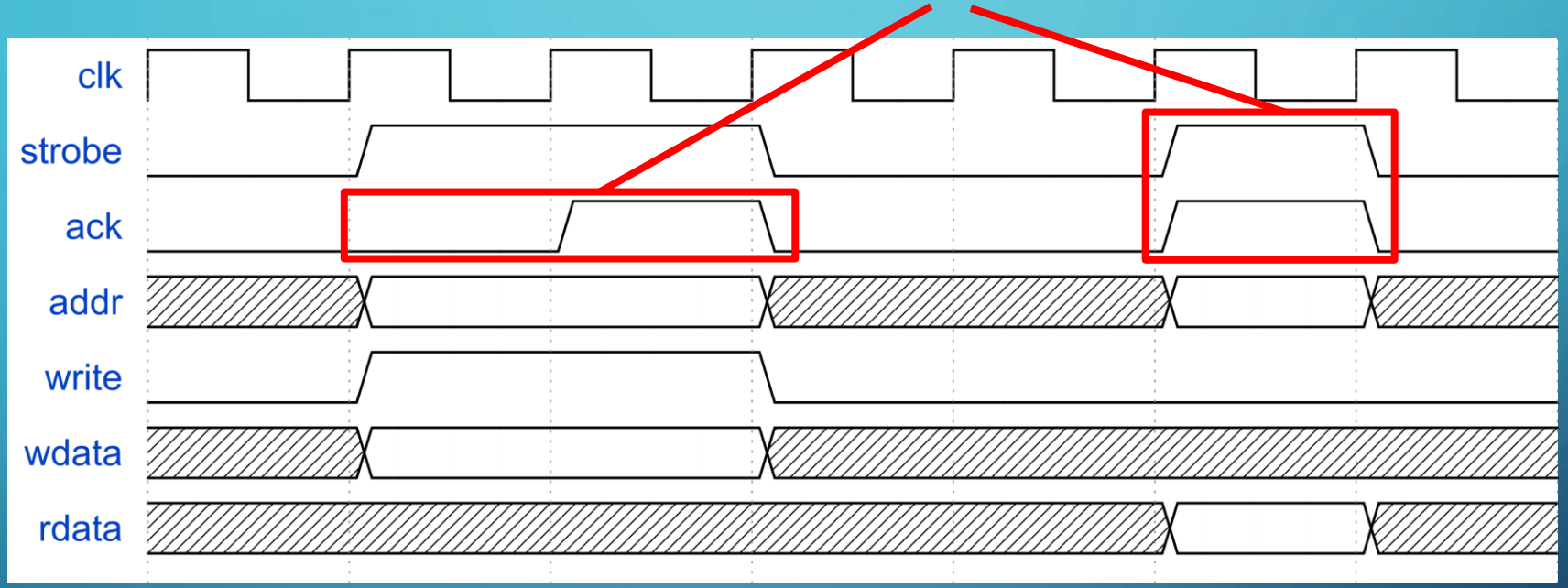
IPBUS

- Slaves signal end of transaction by raising “Acknowledge”



IPBUS

- Slaves signal end of transaction by raising “Acknowledge”
- This may be tied to strobe if the slave responds instantly



IPBUS

- Which means code generally ends up looking like:

```
process(ipb_clk)
begin
if rising_edge(ipb_clk) then
  ack <= '0';
  if ipb_in.ipb_strobe='1' and ack='0' then
    if ipb_in.ipb_write='1' then
      if ipb_in.ipb_addr(...) = Our_addr then
        Some_reg <= ipb_in.ipb_wdata;
        ack <= '1';
      end if;
    else
      if ipb_in.ipb_addr(...) = Our_addr then
        ipb_out.ipb_rdata <= Some_reg;
        ack <= '1';
      end if;
    end if;
  end if;
end if;
end process;
ipb_out.ipb_err <= '0';
ipb_out.ipb_ack <= ack;
```

IPBUS

We only do something when "strobe" is high and "ack" is not

```
process(ipb_clk)
begin
if rising_edge(ipb_clk) then
  ack <= '0';
  if ipb_in.ipb_strobe='1' and ack='0' then
    if ipb_in.ipb_write='1' then
      if ipb_in.ipb_addr(...) = Our_addr then
        Some_reg <= ipb_in.ipb_wdata;
        ack <= '1';
      end if;
    else
      if ipb_in.ipb_addr(...) = Our_addr then
        ipb_out.ipb_rdata <= Some_reg;
        ack <= '1';
      end if;
    end if;
  end if;
end if;
end process;
ipb_out.ipb_err <= '0';
ipb_out.ipb_ack <= ack;
```

IPBUS

If the “write” flag is high and the address is referring to us, we should read data off the bus

```
process(ipb_clk)
begin
if rising_edge(ipb_clk) then
  ack <= '0';
  if ipb_in.ipb_strobe='1' and ack='0' then
    if ipb_in.ipb_write='1' then
      if ipb_in.ipb_addr(...) = Our_addr then
        Some_reg <= ipb_in.ipb_wdata;
        ack <= '1';
      end if;
    else
      if ipb_in.ipb_addr(...) = Our_addr then
        ipb_out.ipb_rdata <= Some_reg;
        ack <= '1';
      end if;
    end if;
  end if;
end if;
end process;
ipb_out.ipb_err <= '0';
ipb_out.ipb_ack <= ack;
```

IPBUS

We set “ack” as soon as we are done with the data that is on the bus

```
process(ipb_clk)
begin
if rising_edge(ipb_clk) then
  ack <= '0';
  if ipb_in.ipb_strobe='1' and ack='0' then
    if ipb_in.ipb_write='1' then
      if ipb_in.ipb_addr(...) = Our_addr then
        Some_reg <= ipb_in.ipb_wdata;
        ack <= '1';
      end if;
    else
      if ipb_in.ipb_addr(...) = Our_addr then
        ipb_out.ipb_rdata <= Some_reg;
        ack <= '1';
      end if;
    end if;
  end if;
end if;
end process;
ipb_out.ipb_err <= '0';
ipb_out.ipb_ack <= ack;
```

IPBUS

Note that an IPbus slave may have multiple addresses for different registers

```
process(ipb_clk)
begin
if rising_edge(ipb_clk) then
  ack <= '0';
  if ipb_in.ipb_strobe='1' and ack='0' then
    if ipb_in.ipb_write='1' then
      if ipb_in.ipb_addr(...) = Our_addr then
        Some_reg <= ipb_in.ipb_wdata;
        ack <= '1';
      end if;
    else
      if ipb_in.ipb_addr(...) = Our_addr then
        ipb_out.ipb_rdata <= Some_reg;
        ack <= '1';
      end if;
    end if;
  end if;
end if;
end process;
ipb_out.ipb_err <= '0';
ipb_out.ipb_ack <= ack;
```

IPBUS

If the “write” flag is low and the address is referring to us, we should put some data onto the bus

```
process(ipb_clk)
begin
if rising_edge(ipb_clk) then
  ack <= '0';
  if ipb_in.ipb_strobe='1' and ack='0' then
    if ipb_in.ipb_write='1' then
      if ipb_in.ipb_addr(...) = Our_addr then
        Some_reg <= ipb_in.ipb_wdata;
         ack <= '1';
      end if;
    else
      if ipb_in.ipb_addr(...) = Our_addr then
         ipb_out.ipb_rdata <= Some_reg;
         ack <= '1';
      end if;
    end if;
  end if;
end if;
end process;
ipb_out.ipb_err <= '0';
ipb_out.ipb_ack <= ack;
```

IPBUS

We set “ack” as soon as there is valid data on the bus
If we are reading from a RAM, this may not be instantly, so we hold “ack” low until we have valid data

```
process(ipb_clk)
begin
if rising_edge(ipb_clk) then
  ack <= '0';
  if ipb_in.ipb_strobe='1' and ack='0' then
    if ipb_in.ipb_write='1' then
      if ipb_in.ipb_addr(...) = Our_addr then
        Some_reg <= ipb_in.ipb_wdata;
         ack <= '1';
      end if;
    else
      if ipb_in.ipb_addr(...) = Our_addr then
         ipb_out.ipb_rdata <= Some_reg;
         ack <= '1';
      end if;
    end if;
  end if;
end if;
end process;
ipb_out.ipb_err <= '0';
ipb_out.ipb_ack <= ack;
```

NOTES ON ADDRESSES

- By default, address-bit 31 is set to ‘1’ for the payload region
  - And ‘0’ for the infrastructure
- But the “magic” behind IPbus means you should never use that bit and never check that bit
- Use as few address bits as you can get away with to conserve resources!

EXERCISE

- Modify your payload to add a user-programmable value to the data presented on each link
  - Simplest: One constant for all links
  - Hardest: Different constant for each link
- Please make your code capable of reading back the constant(s) you set
- Set it building

WRITING THE SAME CODE EACH TIME?

- That doesn't sound like the CMS-UK way

WRITING THE SAME CODE EACH TIME?

- That doesn't sound like the CMS-UK way
- A lot of the time we end up doing very similar tasks
- Copying and pasting code is bad
- Generic out-of-the-box slaves are available in the IPbus Repo

WRITING THE SAME CODE EACH TIME?

- ipbus_reg_v
  - Bank of registers of parameterized size (bus has read & write access to all of them)
- ipbus_roreg_v
  - Block of read-only registers (i.e. values specified at build time, e.g. version numbers)
- ipbus_dpram
  - 32b (or less) wide dual-port memory with IPbus access on one side
- Plus a bunch of more obscure ones

WRITING THE SAME CODE EACH TIME?

- There are also IPbus-interfaces to common bus-protocols
  - SPI
  - I2C
  - JTAG