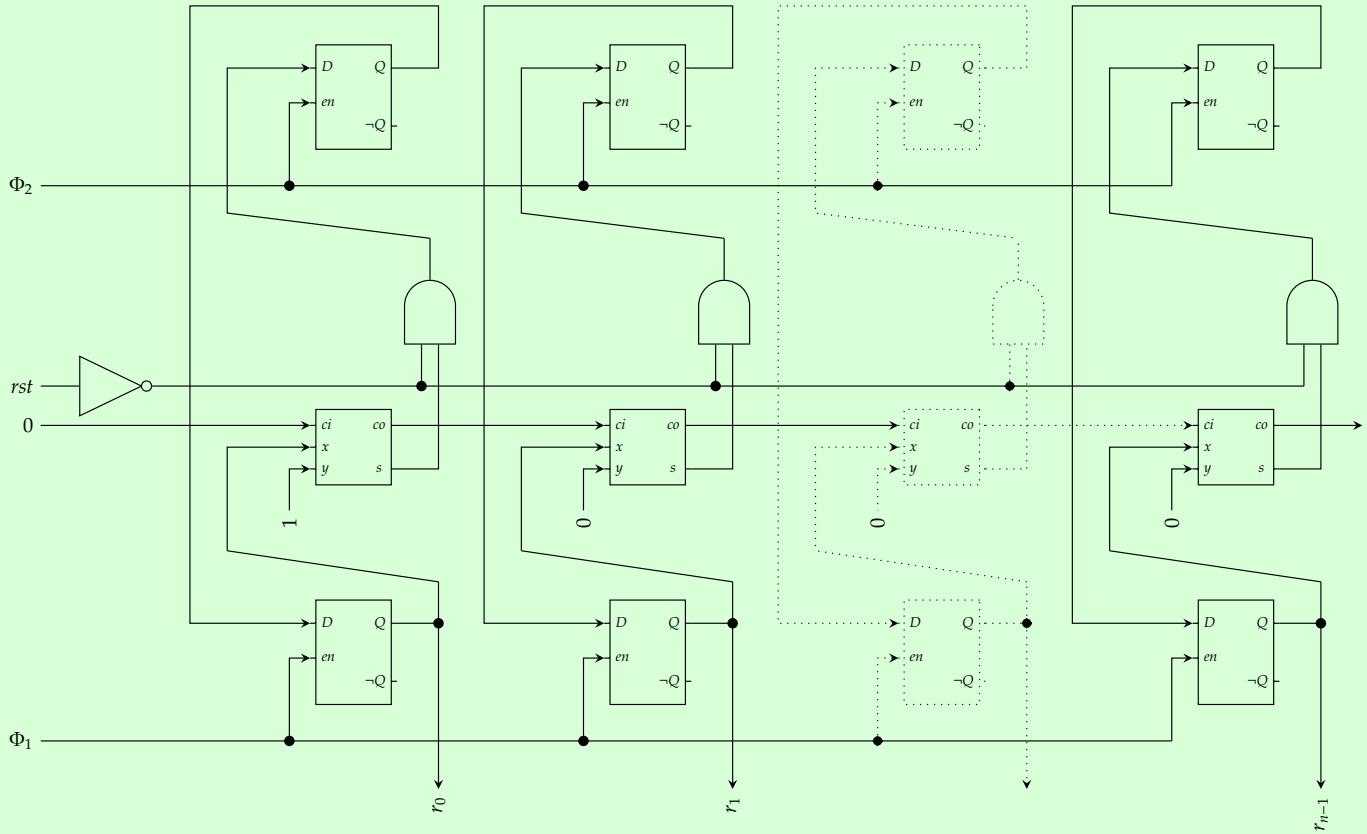
- Recall: we previously designed a cyclic, uncontrolled 4-bit counter, i.e.,

Circuit



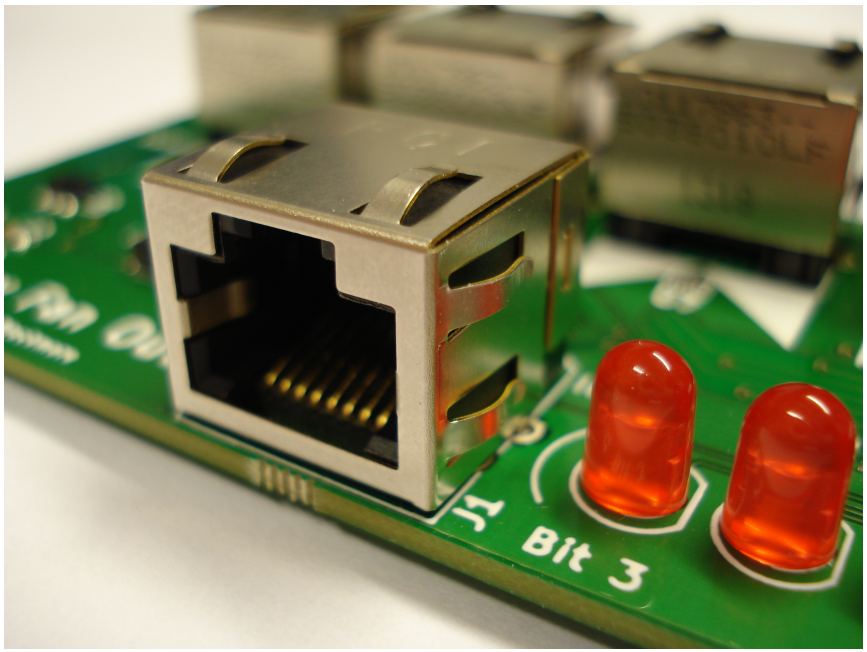
- Question: how can we produce a practical rather than simulated implementation?

- Solution #1: work at a low-level, e.g., via UoB-designed NAND board(s)

1 NOT gate	$\simeq$	$1 \cdot 1$	$=$	1	NAND gates
4 AND gate	$\simeq$	$4 \cdot 2$	$=$	8	NAND gates
8 D-type latches	$\simeq$	$8 \cdot 5$	$=$	40	NAND gates
4 full-adders	$\simeq$	$4 \cdot 9$	$=$	36	NAND gates
			$=$	85	NAND gates
			$\simeq$	6	NAND boards ($+\infty$ wires)

- Solution #2: work at a high(er)-level, e.g.,
  1. via a breadboard plus 78*xx* series digital logic ICs, or
  2. via UoB-designed "build-a-comp" ...

"build-a-comp" (1) – Concept

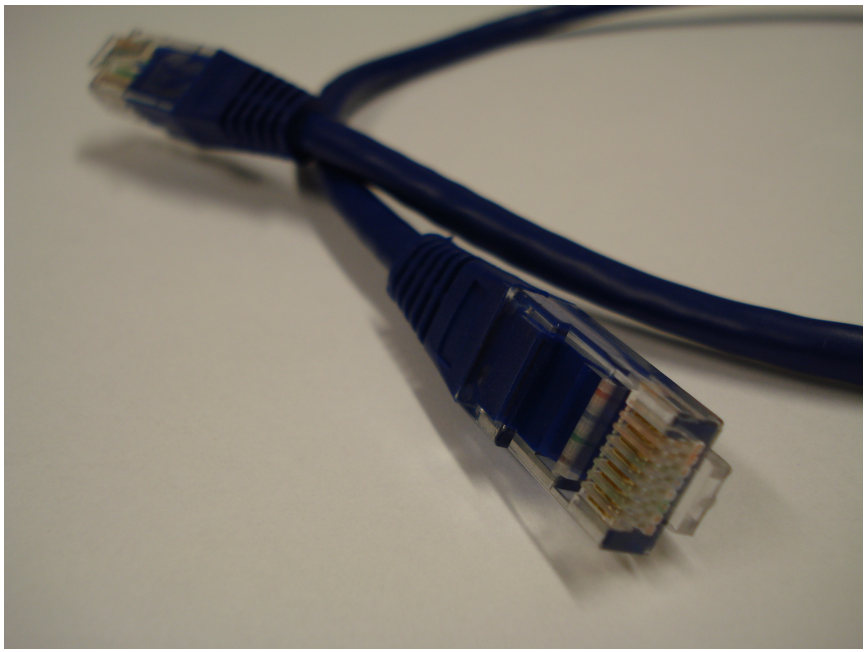


The "build-a-comp" concept combines

1. physical **modules** for (roughly) the same set of components developed thus far,
2. 4-bit control and data ports where data flows bottom-to-top and control right-to-left,
3. ports connected using CAT-5 cable, which also supplies power,
4. unconnected inputs default (i.e., are pulled-down) to 0

plus simulated analogues in Logisim.

"build-a-comp" (1) – Concept



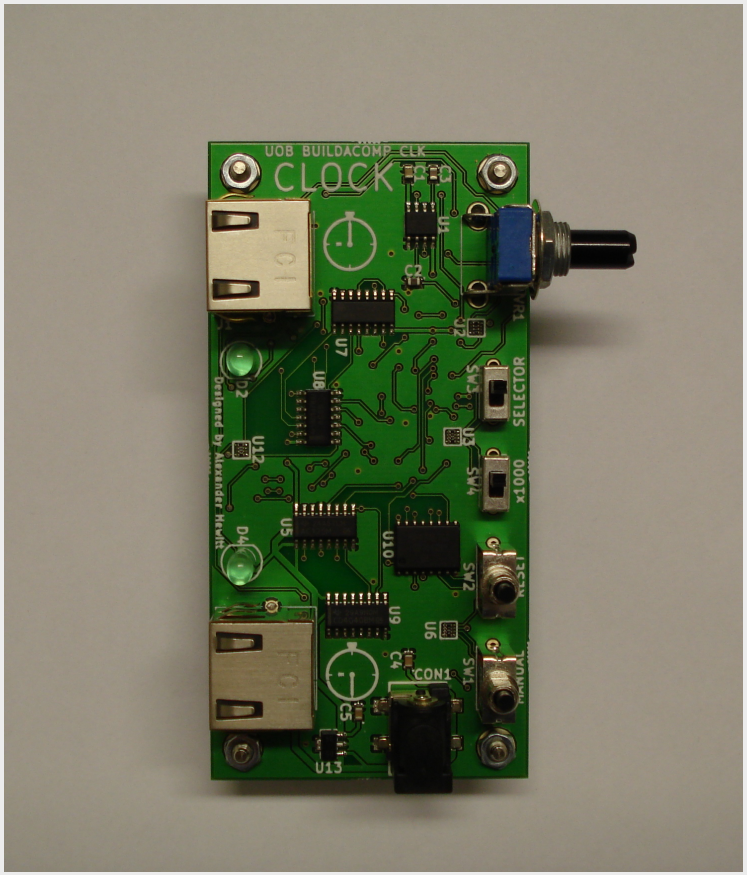
The "build-a-comp" concept combines

1. physical **modules** for (roughly) the same set of components developed thus far,
2. 4-bit control and data ports where data flows bottom-to-top and control right-to-left,
3. ports connected using CAT-5 cable, which also supplies power,
4. unconnected inputs default (i.e., are pulled-down) to 0

plus simulated analogues in Logisim.

"build-a-comp" (2) – Examples

Module



Quote

The clock module produces a two-phase clock. The phases do not overlap. It has two outputs a and b:

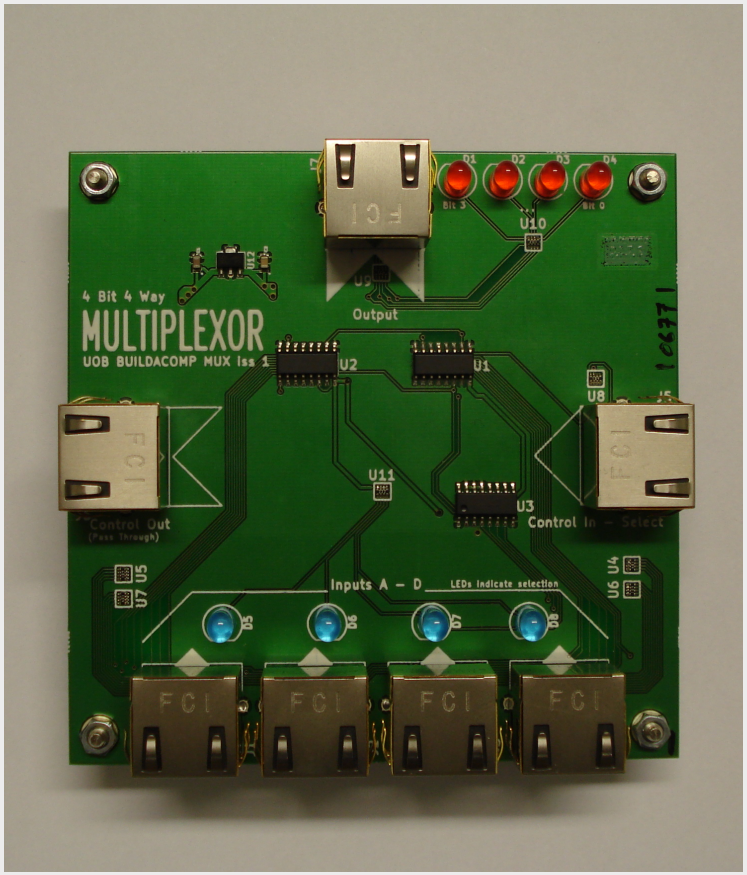
a_0 : *clock phase* 1	b_0 : *clock phase* 2
a_1 : *reset*	b_1 : *reset*
a_2 : *high*	b_2 : *high*
a_3 : *not used*	b_3 : *not used*

The module can be set by switches to select between a high speed crystal clock and a low speed clock. The low-speed clock can be varied between 0.1 *Hz and* 100 *Hz. The module has a reset button that activates the reset signal between the end of phase* 2 *and the start of phase* 1*.*

– build-a-comp specification

"build-a-comp" (3) – Examples

Module



Quote

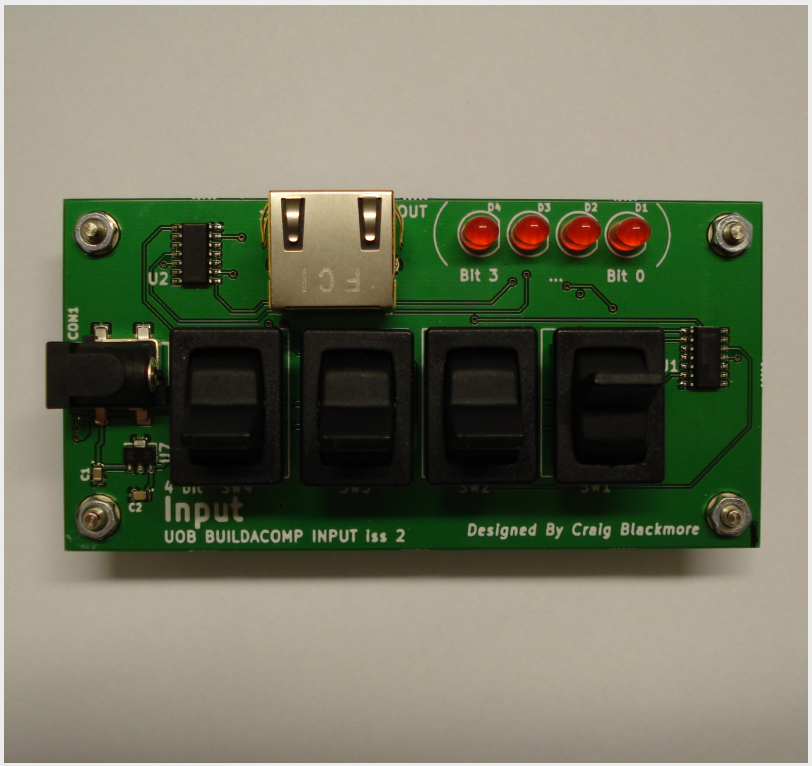
The multiplexor module selects one of four data inputs and outputs the selected data on a single data output. Its control signals are

c_0 *: input select*
c_1 *: input select*
c_2 *: not used*
c_3 *: not used*

– build-a-comp specification

"build-a-comp" (4) – Examples

Module



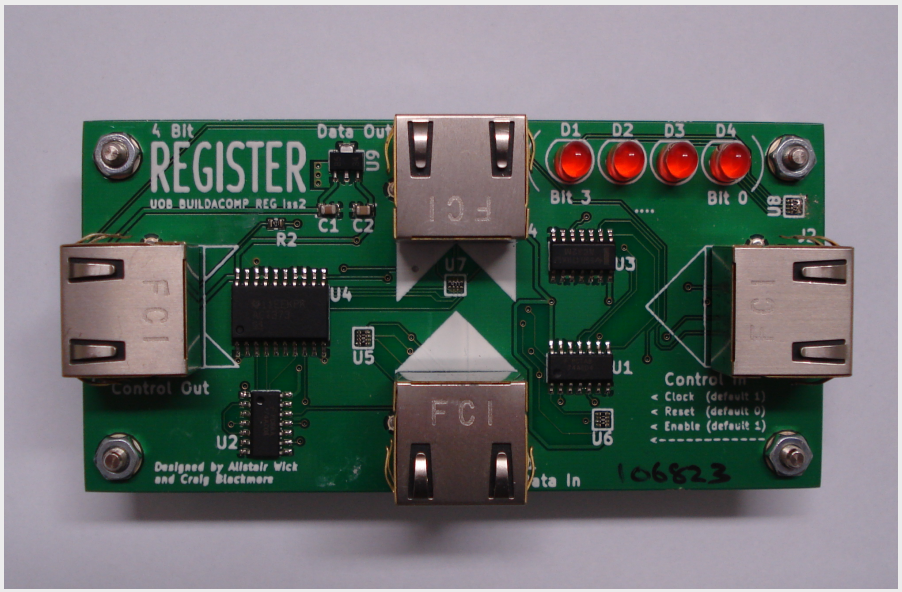
Quote

The input module has four switches and a single four bit output.

– build-a-comp specification

"build-a-comp" (5) – Examples

Module



Quote

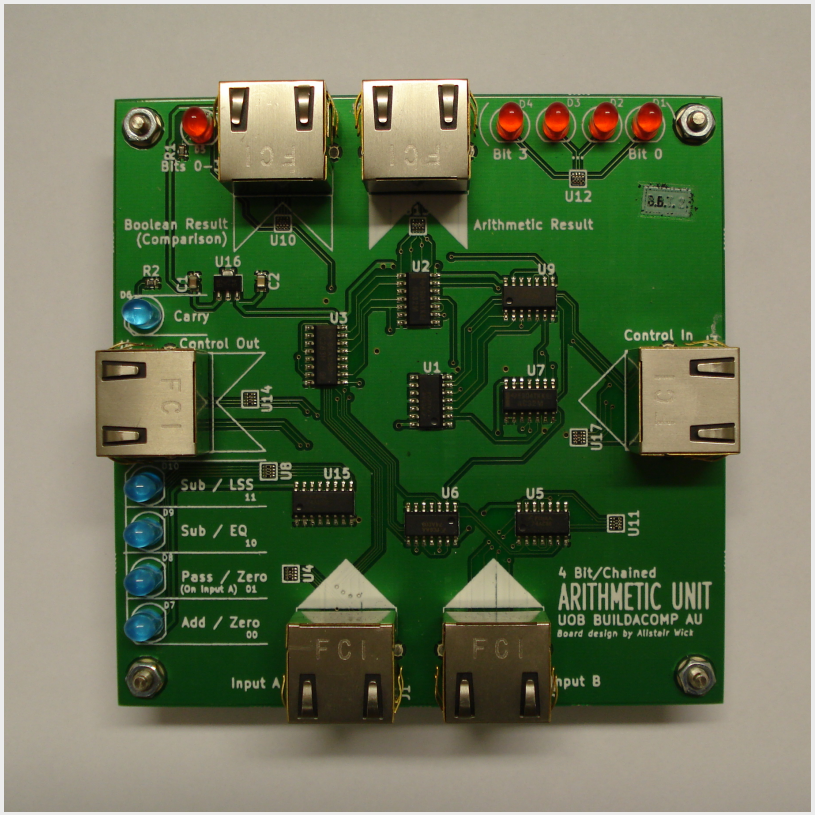
The register module holds data. It has an enable input and a clock input. Data is stored in the register when both the enable signal and the clock signal are active; the state of the data output follows the data input and is held after the clock signal becomes inactive. The register can be set to 0 using a reset signal. The control signals are

c_0 : *clock*
c_1 : *reset*
c_2 : *enable*
c_3 : *not used*

– build-a-comp specification

"build-a-comp" (6) – Examples

Module



Quote

The arithmetic module has two data inputs and a data output. It performs addition and subtraction, and can also simply pass one of its operands to its data output. It also has a Boolean data output that reflects the state of its output (zero or negative). The Boolean output has all four bits set (true) or all four bits clear (false). The module has a control carry input and produces a control carry output. It also has a control zero-detect input and produces a control zero-detect output. The control signals, and their effect on the data and Boolean outputs are as follows:

c_0 *: function select*
c_1 *: function select*
c_2 *: carry in and carry out*
c_3 *: not-zero in and not-zero out*

This means that if no control input is connected, it will add its operands and also reflect the result (zero or non-zero) on its Boolean output. Also, if one data input is not connected, the default behaviour will provide zero-detection.

– build-a-comp specification

Demo and discussion

Conclusions

- Take away points:
  1. The unit introduction motivated the use of abstraction; for example, the limited HP-35 calculator depended on
     - Finite State Machines (FSMs) and arithmetic designs (e.g., ripple-carry adder), which depend on
     - combinatorial (e.g., multiplexer, full-adder) and sequential (e.g., latch) components, which depend on
     - logic gates, which depend on
     - transistors (or switches more generally), which depend on
     - fundamental Physical and Mathematics.
  2. This is good: it allows us to manage scale and complexity, e.g., by using abstract modules once their internal components and design are understood.
  3. This is bad: you *really* need to understand one layer before moving upward, implying a need to *thoroughly* revise material from this first part *now*!