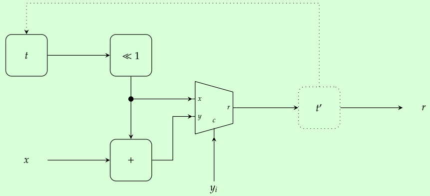
- Recall: our goal is to implement a bit-serial multiplier, i.e.,

Algorithm

Input: Two unsigned, n-bit, base-2 integers x and y
Output: An unsigned, $2n$-bit, base-2 integer $r = y \cdot x$

```
t ← 0
for i = n − 1 downto 0 step −1 do
    t ← 2 · t
    if y_i = 1 then
        t ← t + x
    end
end
return t
```

Circuit



as a case-study of data- and control-paths; we more or less have the data-path, but what about the control-path ...

Question

Design an FSM-based component that replicates the behaviour of a loop counter, for example `i` within a C-style `for` loop such as

```
for( int i = m; i < n; i++ ) {
  ...
}
```

noting that

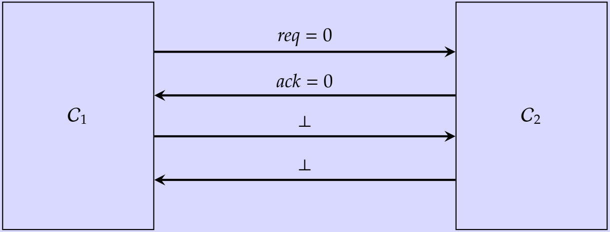
1. like C, we'll allow the loop counter `i` to equal `n` once the loop is complete,
2. we'll need a mechanism that informs us when this is, plus also allows us to start iteration, and
3. we'll look at *a* solution, not *all* solutions.

An Aside: A simple, generic control protocol

- Question: given a user C_1 of some component C_2, how does
  1. C_2 know when to start computation (e.g., when any input x is available), and
  2. C_1 know when computation has finished (e.g., when any output $r = f(x)$ is available).
- Solution(s):
  1. use a shared clock signal to synchronise events somehow, or
  2. use a simple **control protocol** based on two signals
     2.1 *req* (or **request**), and
     2.2 *ack* (or **acknowledge**).

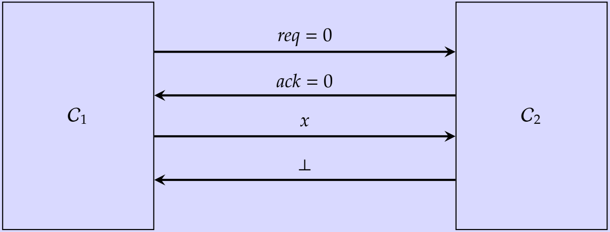
An Aside: A simple, generic control protocol

Algorithm



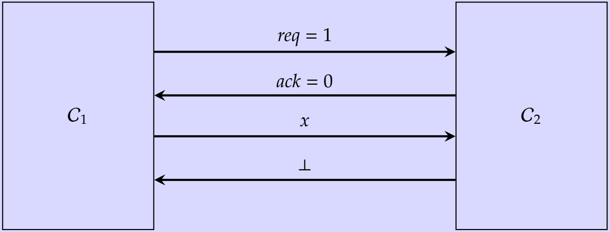
An Aside: A simple, generic control protocol

Algorithm



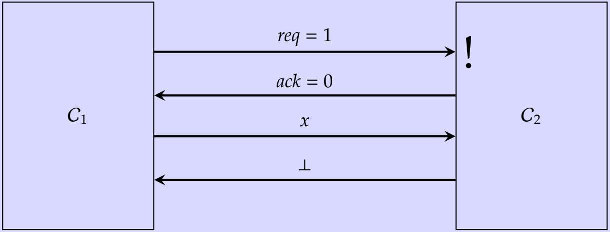
An Aside: A simple, generic control protocol

Algorithm



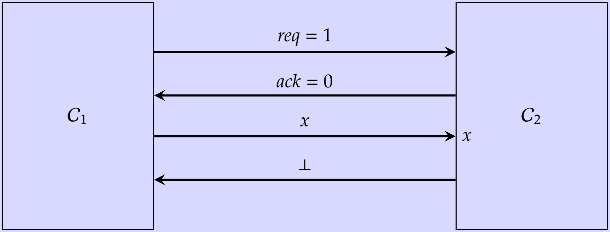
An Aside: A simple, generic control protocol

Algorithm



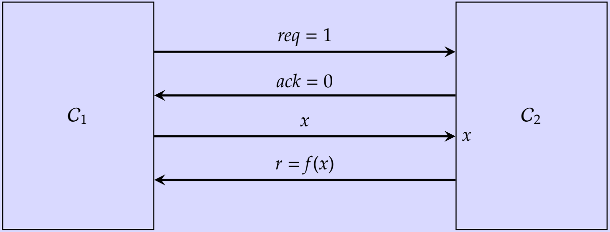
An Aside: A simple, generic control protocol

Algorithm



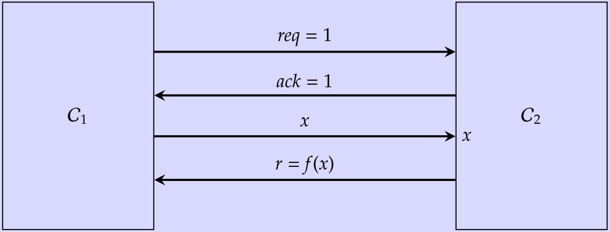
An Aside: A simple, generic control protocol

Algorithm



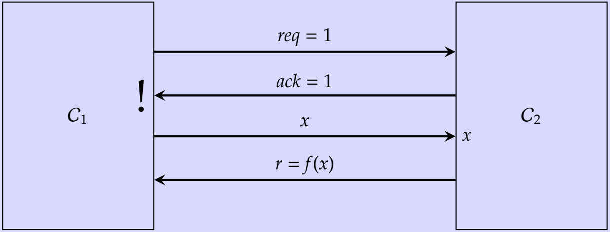
An Aside: A simple, generic control protocol

Algorithm



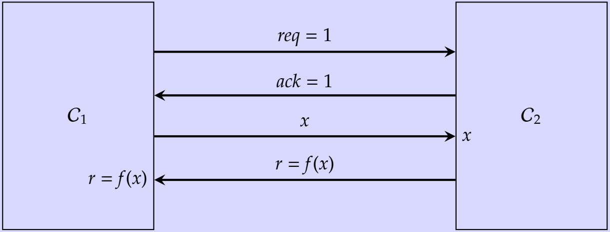
An Aside: A simple, generic control protocol

Algorithm



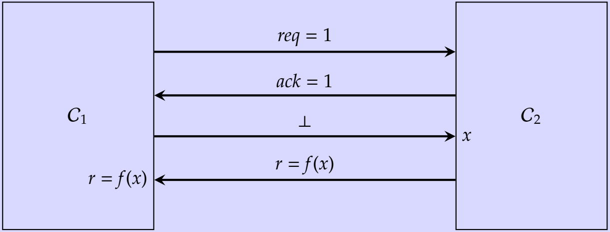
An Aside: A simple, generic control protocol

Algorithm



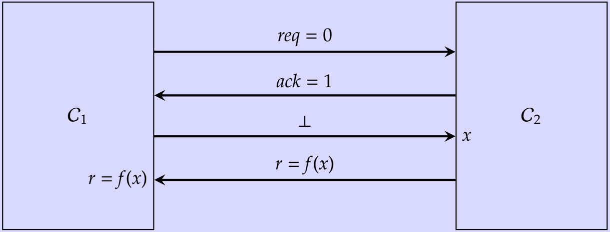
An Aside: A simple, generic control protocol

Algorithm



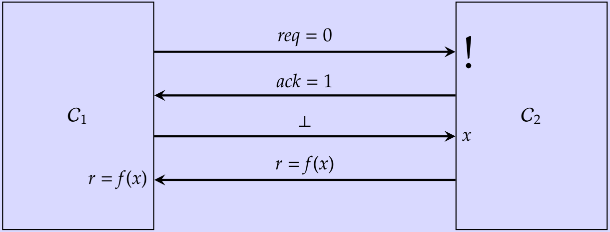
An Aside: A simple, generic control protocol

Algorithm



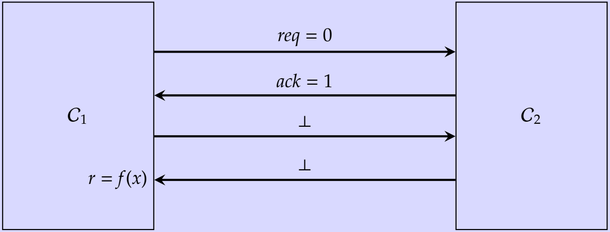
An Aside: A simple, generic control protocol

Algorithm



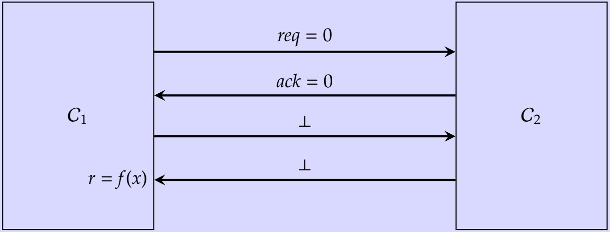
An Aside: A simple, generic control protocol

Algorithm



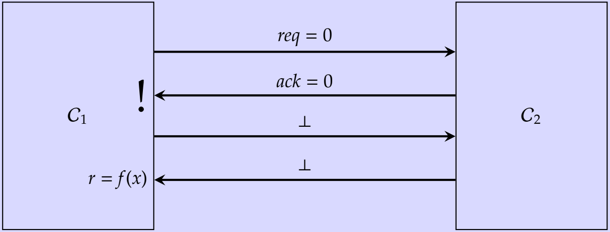
An Aside: A simple, generic control protocol

Algorithm



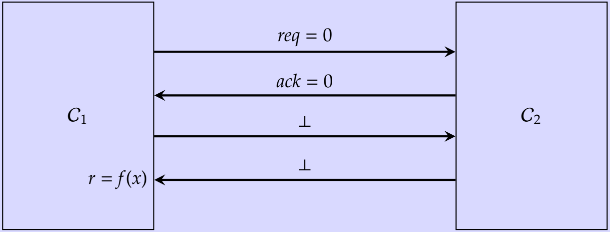
An Aside: A simple, generic control protocol

Algorithm



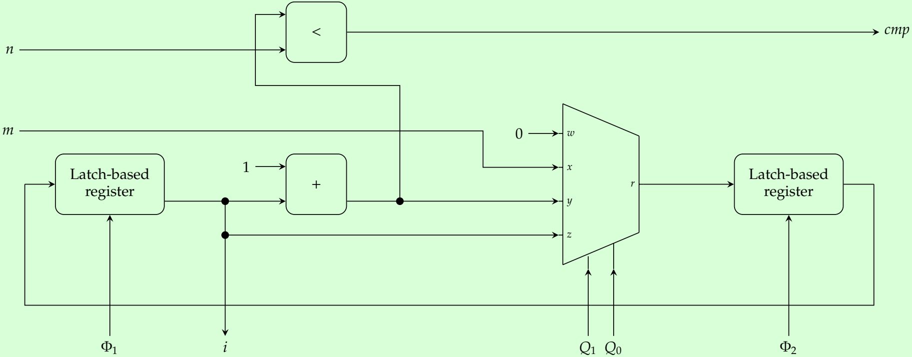
An Aside: A simple, generic control protocol

Algorithm



A controlled “loop counter” component (1)

Circuit (data-path, sketch)



A controlled "loop counter" component (2)

- Our FSM can be in one of 4 states:
  - in S_{wait} it waits for a request (i.e., for $req = 1$),
  - in S_{init} it uses any input to initialise itself (e.g., setting the initial loop counter value),
  - in S_{step} it performs an iteration of the loop, and
  - in S_{done} it waits for $req = 0$ (while setting $ack = 1$) once the loop is complete.
- Since $2^2 = 4$, we can assign a concrete 2-bit value

$$
\begin{array}{lcl}
S_{wait} & \mapsto & \langle 0, 0 \rangle \\
S_{init} & \mapsto & \langle 1, 0 \rangle \\
S_{step} & \mapsto & \langle 0, 1 \rangle \\
S_{done} & \mapsto & \langle 1, 1 \rangle
\end{array}
$$

  to each abstract label; this basically means we can talk about

  1. $Q = \langle Q_0, Q_1 \rangle$ as being the current state, and
  2. $Q' = \langle Q'_0, Q'_1 \rangle$ as being the next state.

A controlled "loop counter" component (3)

Algorithm (control-path, tabular)

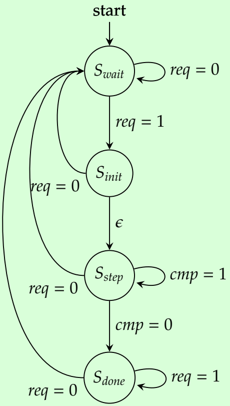
Algorithm (control-path, diagram)

A controlled "loop counter" component (3)

Algorithm (control-path, tabular)

	Q	δ: Q', $cmp = 0$	δ: Q', $cmp = 1$	ω: ack, $cmp = 0$	ω: ack, $cmp = 1$
$req = 0$	S_{wait}	S_{wait}	S_{wait}	0	0
	S_{init}	S_{wait}	S_{wait}	0	0
	S_{step}	S_{wait}	S_{wait}	0	0
	S_{done}	S_{wait}	S_{wait}	1	1
$req = 1$	S_{wait}	S_{init}	S_{init}	0	0
	S_{init}	S_{step}	S_{step}	0	0
	S_{step}	S_{done}	S_{step}	0	0
	S_{done}	S_{done}	S_{done}	1	1

Algorithm (control-path, diagram)



A controlled "loop counter" component (4)

Algorithm (control-path, truth table)

Rewriting the abstract labels yields the following concrete truth table:

				δ		ω
req	cmp	Q_1	Q_0	Q'_1	Q'_0	ack
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	1	0	0
1	0	1	0	1	1	0
1	0	1	1	1	1	1
1	1	0	0	0	1	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	1	1	1

A controlled "loop counter" component (5)

Circuit (control-path, δ)

Translating the truth table into a set of Karnaugh maps

yields the following Boolean expressions:

Circuit (control-path, δ)

Translating the truth table into a set of Karnaugh maps

Q_1'

0	0	0	0
0	0	0	0
0	1	1	1
0	1	1	1

Q_0'

0	0	0	0
0	0	0	0
1	0	1	0
1	0	1	1

yields the following Boolean expressions:

$$
\begin{array}{rcl}
Q_1' & = & (\; req \; \wedge \; Q_0 \;) \; \vee \\
 & & (\; req \; \wedge \; Q_1 \;) \\
 & & \\
Q_0' & = & (\; req \; \wedge \; \neg Q_1 \; \wedge \; \neg Q_0 \;) \; \vee \\
 & & (\; req \; \wedge \; \wedge \; Q_1 \; \wedge \; Q_0 \;) \; \vee \\
 & & (\; req \; \wedge \; \neg cmp \; \wedge \; Q_1 \;)
\end{array}
$$

A controlled "loop counter" component (6)

Circuit (control-path, ω)

Translating the truth table into a set of Karnaugh maps

yields the following Boolean expressions:

A controlled "loop counter" component (6)

Circuit (control-path, ω)

Translating the truth table into a set of Karnaugh maps

ack (Q_1, Q_0 across; *cmp*, *req* down)

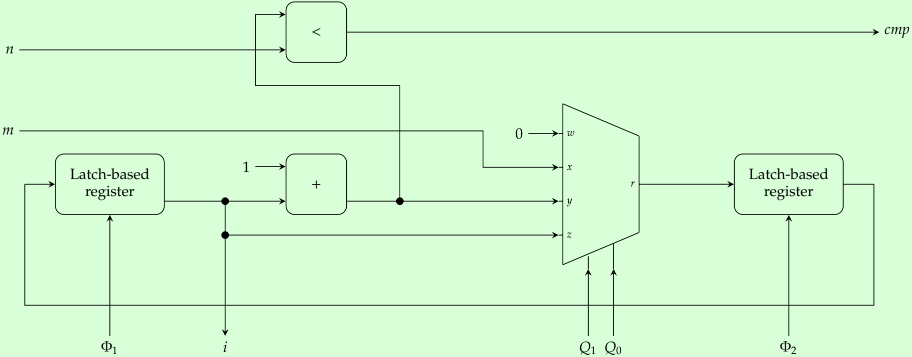
0	0	1	0
0	0	1	0
0	0	1	0
0	0	1	0

yields the following Boolean expressions:

$$ack = Q_1 \wedge Q_0$$

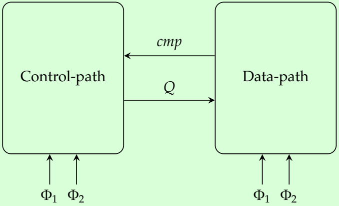
A controlled "loop counter" component (7)

Circuit (data-path, finalised)



A controlled "loop counter" component (8)

Circuit (data- and control-paths)



Demo and discussion

Conclusions

- Next steps (or, the lab. session):
  - We now have the loop counter component implemented as specified ...
  - ... the next challenge is clearly then *using* it to realise the original goal, e.g., specifying
    1. any additional data-path components required, and
    2. how loop counter (the control-path) controls them

    so we end up with a bit-serial multiplier.