Digital Electronics (Part 2)

UNIVERSITY OF
CAMBRIDGE

Computer Laboratory

Part Ia, Part II (General) and Diploma in Computer Science

Contents

Lecture 9 — LSI components and memories

Contents:

- 74HC series chips
- SRAM
- address, data and control signals
- tristate

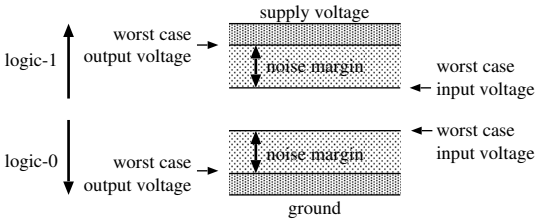
Example: 74HC193 counter chip

- referring to 74HC193 datasheet, note:
  - general description
  - pin out & pin definitions
  - function table
  - timing information
  - parallel load function

HCMOS family characteristics

- referring to HCMOS family datasheet pages 2 & 4:
  - power supply tolerances
  - valid input/output voltages and noise margins

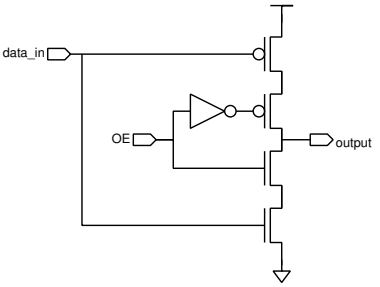
Noise Margins



Memories

- store data in a volatile way (e.g. SRAM) or non volatile (e.g. FLASH, EEPROM, ROM)
- signals:
  - address bus — n bits which specify which location in memory is being referred to
  - data bus — m bits of bidirectional tristate bus transmitting data to be read/written
  - $\overline{\text{WE}}$ — not write enable — determines whether data is to be read (1) or written (0)
  - $\overline{\text{OE}}$ — not output enable — determines if data is to be output (0) or not (1)
  - $\overline{\text{CS}}$ — not chip select — determines if the chip is to be activated (0) or not (1).

Tristate drivers



- when OE (output enable) is low the output is tristate

Datasheet: HT6116 SRAM

- referring to the HT6116 datasheet:
  - address and data buses
  - control signals
  - timing

Supervision Work

- design a 3–of–8 address decoder (i.e. 3 bits of address which decode to 8 word lines)
- design the state machine for workshop 4

Lecture 10 – Programmable Logic Devices

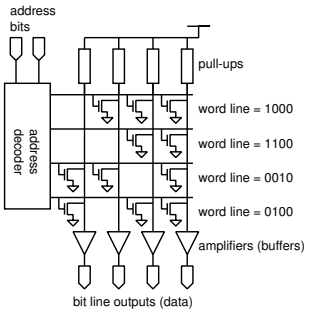
Contents:

- ROMs
- PALs
- FPGAs

ROMs – Overview

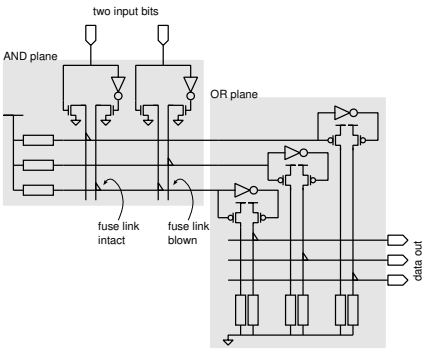
- ROM = Read Only Memory
- implements a table lookup where an *address* specifies a row in the table which holds the *data*
- for n bits of address we need 2^n rows in our table
- the n bits of address are decoded to 2^n *word lines*
- for m bit wide data the, size of a ROM $=O(m \times 2^n)$

ROMs – array structure



Notes:
2 address bits
4 data bits
so 2^2x4 = 16 bits

PALs – basic structure



PALs – notes on datasheet

- page 1 shows the top level functionality and chip pin layout
- page 3 describes the Output Logic Macrocell which can emulate a wide range of old PAL devices
- page 4 shows the fixed OR plane, with our without DFF
- page 5 shows the extensive AND array structure which means that the fixed OR structure does not limit functionality
- pages 6–9 describe other modes (removed from handout)
- pages 11–13 contain the timing information (removed from handout)

PALs vs ROMs

- scaling issues:
  - ROM size is proportional to $O(2^{\text{address}-\text{bits}})$
  - PALs size is proportional to the number of terms that can be defined

FPGAs

- FPGA = Field Programmable Gate Array
- contains a sea of programmable wiring and function units controlled by memory bits which hold the program
- function units are typically 4 input, 1 output lookup tables with an optional DFF on the output
- more in Structured Hardware Design and ECAD

Supervision Work

- design determine the PAL equations for the state machines in workshops 2 & 3

Lecture 11 — Asynchronous Logic

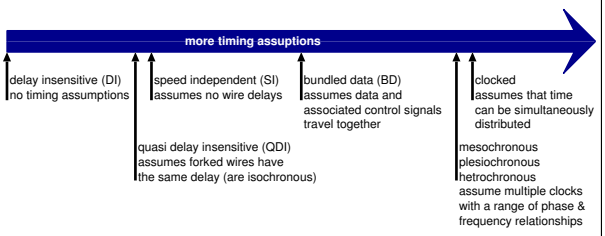
Contents:

- asynchronous FIFO buffer design
- signal transition graphs
- C-elements

Synchronous vs asynchronous circuits

- Synchronous (clocked) circuits
  - discrete time provided by the clock
  - state is usually held in clocked flip-flops
  - typically no loops in combinational logic (no feedback)
  - easy to design in the small but can be problematic in the large
- Asynchronous circuits
  - continuous time
  - state is held in none clocked flip-flops
  - state holding elements often formed from combinational logic with loops (logic with feedback)
  - complex to design in the small but composes simply in the large

Fairbanks Scale: timing assumptions

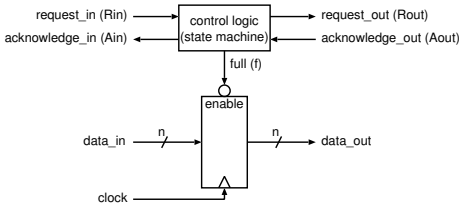


Data signalling

- communication often required between a producer and a consumer which may be run from the same clock but produce/consume data at different rates
- signalling required for flow control to avoid data loss
- *delay insensitive* signalling guarantees that the consumer has consumed data before the producer produces more data
  - i.e. the producer waits until it has been acknowledged by the consumer
  - useful for synchronous design
  - vital for asynchronous design since there is no other mechanism (e.g. a clock) to ensure data ordering

Example problem: Synchronous FIFO

- FIFO (First In First Out) buffers are often used to provide decoupled communication
- assume we have the following basic one place buffer arrangement:

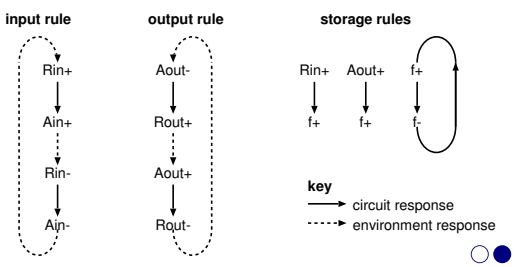


Sync. FIFO: Environment rules

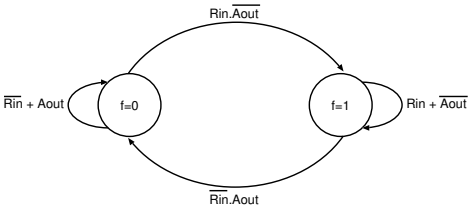
- we will use 4-phase level signalling on the input:
  - if Ain=0 then at some point Rin may go high
  - when Rin=1 then we expect Ain to go high eventually
  - if Ain=1 then at some point Rin may go low
  - when Rin=0 then we expect Ain to go low eventually
- storage assumption: when Ain=1 we assume that the data has been latched
- similarly on the output side

Signal Transition Graph Snippets

- STGs specify the signal ordering rather than particular states
- STG snippets can specify requirements



Sync. FIFO: State transition graph

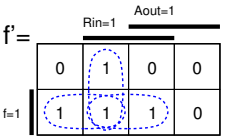


- note that Ain=Rout=f in this example

Sync. FIFO: State table

current state			next state
f	Rin	Aout	f'=Ain=Rout
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

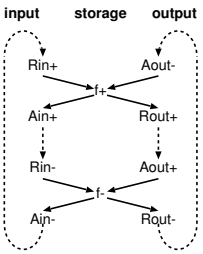
Sync. FIFO: K-map for f



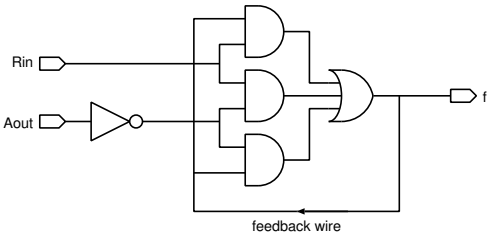
- $f' = Rin.\overline{Aout} + f.Rin + f.\overline{Rout}$
- note:
  - glitch free function of f' if just one input changes at a time
  - the environment rules specify that just one input changes at a time
  - so do we need a DFF between f' and f?

Sync to Async

- the signal transition graph for our f asynchronous state machine is below
- this satisfies the STG snippets



Gate level implementation



Muller C-element

- the function f we've just created is actually a Muller C-element with one input inverted
- the C-element is a little known latch designed by Muller in the 1950s
- truth table:

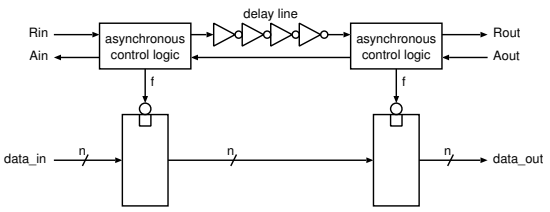
inputs		output
a	b	c'
0	0	0
0	1	c
1	0	c
1	1	1

RS latch implementations

- we could have implemented f using an RS latch with the following input functions:
  $S = Rin.\overline{Aout}$
  $R = \overline{Rin}.Aout$
- the S input function determines the input combinations in the *on-set*
- the R input function determines the input combinations in the *off-set*
- for a valid asynchronous state machine:
  - the input transitions must only respond when the state is appropriate
  - the on-set and off-set should be disjoint and be implemented with glitch free functions

Timing assumptions for async. FIFO

- the data must arrive before the request
- delay-lines (e.g. chains of inverters) are typically added in the request (Rout to Rin)



Supervision Work

- if n asynchronous FIFO storage elements from this lecture were connected together, how many data items could we store?
- hard problem: how could we improve the FIFO storage?
- suggested exam. questions useful for a revision supervision:
  2001 Paper 2 Questions 2 and 3
  2001 Paper 10 Question 1 parts a to e
  2002 Paper 2 Question 2
  2002 Paper 10 Question 1
  2003 Paper 2 Questions 2 and 3
  2003 Paper 10 Question 1