Key Points addressed in this Topic

- Taxonomy of Memories
- Memory organisation
- Static memory timing
- Embedded RAM in Virtex FPGA – dual port RAM
- SRAM application - FIFO
- Dynamic memory
- Synchronous DRAM (SDRAM)
- SDRAM Timing

Slides based on notes from Paolo Ienne, EPFL & David Cullen, Berkeley.
Register File

- Register file from microprocessor

![Diagram of register file with inputs regid and WE, and outputs IN and OUT.]

\[
\text{regid} = \text{register identifier (address of word in memory)}
\]
\[
\text{sizeof(regid)} = \log_2(\# \text{ of reg})
\]
\[
\text{WE} = \text{write enable}
\]

Register File Internals

- For read operations, functionally the register file is equivalent to a 2-D array of flip-flops with tristate outputs on each
  - MUX, but distributed
  - Unary control

- Cell with added write logic:

![Diagram of register file internals with inputs WE and outputs sel_reg0, sel_reg1, FFs, and bit line.]

Regid (address) Decoding

- The function of the address decoder is to generate a one-hot code word from the address.
  - Binary -> unary
  - Simplified DEMUX

- The output is used for row selection.
  - Many different circuits exist for this function. A simple one is shown.
  - Where have you seen this before?

Accessing Register Files

- Read: output is a combinational function of the address input
- Write is synchronous
  - If enabled, input data is written to selected word on the clock edge
- Often multi-ported

![Diagram of accessing register files with inputs clk, addr, dout, din, and WE.]
Simplified RAM organization

Different internal array organization

Typical organisation for a 4K x 16 bit RAM

Typical SRAM Timing
Memory Blocks in FPGAs

- LUTs can double as small RAM blocks:
- Newer FPGA families include larger on-chip RAM blocks (usually dual ported):
  - Called block selectRAMs in Xilinx Virtex series
  - 4k bits each

![RAMB4_S#](image)

Synchronous SRAM timing

Virtex “Block RAMs”

- Each block SelectRAM (block RAM) is a fully synchronous (synchronous write and read) dual-ported (true dual port) 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured independently, providing built-in bus-width conversion.
- CLKA and CLKB can be independent, providing an easy way to “cross clock boundaries”.
- Around 160 of these on the XCV2000E. Multiples can be combined to implement, wider or deeper memories.

Dual-ported Memory Internals

- Add decoder, another set of read/write logic, bits lines, word lines:
- Example cell: SRAM
- Repeat everything but cross-coupled inverters.
- This scheme extends up to a couple more ports, then need to add additional transistors.
First-in-first-out (FIFO) Memory

- Used to implement queues.
- These find common use in computers and communication circuits.
- Generally, used for rate matching data producer and consumer:
  - Producer can perform many writes without consumer performing any reads (or vice versa). However, because of finite buffer size, on average, need equal number of reads and writes.
- Typical uses:
  - interfacing I/O devices. Example network interface. Data bursts from network, then processor bursts to memory buffer (or reads one word at a time from interface). Operations not synchronized.
  - Example: Audio output. Processor produces output samples in bursts (during process swap-in time). Audio DAC clocks it out at constant sample rate.

FIFO Interfaces

- Address pointers are used internally to keep next write position and next read position into a dual-port memory.
- If pointers equal after write $\Rightarrow$ FULL:
  - Address pointers are used internally to keep next write position and next read position into a dual-port memory.
- If pointers equal after read $\Rightarrow$ EMPTY:

FIFO Implementation

- Assume, dual-port memory with asynchronous read, synchronous write.
- Binary counter for each of read and write address. CEs controlled by WE and RE.
- Equal comparator to see when pointers match.
- Flip-flop each for FULL and EMPTY flags:

<table>
<thead>
<tr>
<th>WE</th>
<th>RE</th>
<th>equal</th>
<th>EMPTY</th>
<th>FULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 0</td>
<td>1 0</td>
<td>EMPTY</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>EMPTY</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 0</td>
<td>EMPTY</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>EMPTY</td>
<td>FULL</td>
<td>1</td>
</tr>
</tbody>
</table>

SRAM vs DRAM

- Completely different storage mechanism:
  - Capacitor discharges (leakage) and needs refresh every some time ("seldom", i.e. typ. ~10ms)
  - Additional design complexity (sometimes hidden from the designer, esp. in ASIC)
Classical DRAM Organization (Square)

- bit (data) lines
- Each intersection represents a 1-T DRAM Cell
- Square keeps the wires short: Power and speed advantages; Less RC, faster precharge and discharge is faster access time!

- Row and Column Address together select 1 bit a time

DRAM Logical Organization (4 Mbit)

- 4 Mbit = 22 address bits
- 11 row address bits
- 11 col address bits

- Square root of bits per RAS/CAS
  - Row selects 1 row of 2048 bits from 2048 rows
  - Col selects 1 bit out of 2048 bits in such a row

Logic Diagram of a Typical DRAM

- Control Signals (RAS_L, CAS_L, WE_L, OE_L) are all active low
- Din and Dout are combined (D):
  - WE_L is asserted (Low), OE_L is disasserted (High)
    - D serves as the data input pin
  - WE_L is disasserted (High), OE_L is asserted (Low)
    - D is the data output pin
- Row and column addresses share the same pins (A)
  - RAS_L goes low: Pins A are latched in as row address
  - CAS_L goes low: Pins A are latched in as column address
  - RAS/CAS edge-sensitive

Basic DRAM read & write

- Strobe address in two steps
**DRAM READ Timing**

- Every DRAM access begins at:
  - Assertion of the RAS_L
  - 2 ways to read: early or late v. CAS

**DRAM WRITE Timing**

- Every DRAM access begins at:
  - The assertion of the RAS_L
  - 2 ways to write: early or late v. CAS

---

**Key DRAM Timing Parameters**

- \( t_{RAC} \): minimum time from RAS line falling to the valid data output.
  - Quoted as the speed of a DRAM
  - A fast 4Mb DRAM \( t_{RAC} = 60 \text{ ns} \)
- \( t_{RC} \): minimum time from the start of one row access to the start of the next.
  - \( t_{RC} = 110 \text{ ns} \) for a 4Mbit DRAM with a \( t_{RAC} \) of 60 ns
- \( t_{CAC} \): minimum time from CAS line falling to valid data output.
  - 15 ns for a 4Mbit DRAM with a \( t_{RAC} \) of 60 ns
- \( t_{PC} \): minimum time from the start of one column access to the start of the next.
  - 35 ns for a 4Mbit DRAM with a \( t_{RAC} \) of 60 ns

---

**Memory in Desktop Computer Systems:**

- **SRAM** (lower density, higher speed) used in CPU register file, on- and off-chip caches.
- **DRAM** (higher density, lower speed) used in main memory

**Closing the GAP:**
1. Caches are growing in size.
2. Innovation targeted towards higher bandwidth for memory systems:
   - SDRAM - synchronous DRAM
   - RDRAM - Rambus DRAM
   - EDO RAM - extended data out SRAM
   - Three-dimensional RAM
   - hyper-page mode DRAM video RAM
   - multibank DRAM
DRAM with Column buffer

- Column Latches
- Sense Amps
- Memory Array (2,048 x 2,048)
- Word Line
- Storage Cell

Pull column into fast buffer storage
Access sequence of bit from there

Optimized Access to Cols in Row

- Often want to access a sequence of bits
- Page mode
  - After RAS / CAS, can access additional bits in the row by changing column address and strobing CAS
- Static Column mode
  - Change column address (without repeated CAS) to get different bit
- Nibble mode
  - Pulsing CAS gives next bit mod 4
- Video ram
  - Serial access

SDRAM Details

- Multiple “banks” of cell arrays are used to reduce access time:
  - Each bank is 4K rows by 512 “columns” by 16 bits (for our part)
- Read and Write operations as split into RAS (row access) followed by CAS (column access)
- These operations are controlled by sending commands
  - Commands are sent using the RAS, CAS, CS, & WE pins.
- Address pins are “time multiplexed”
  - During RAS operation, address lines select the bank and row
  - During CAS operation, address lines select the column.
- “ACTIVE” command “opens” a row for operation
  - Transfers the contents of the entire to a row buffer
- Subsequent “READ” or “WRITE” commands modify the contents of the row buffer.
  - For burst reads and writes during “READ” or “WRITE” the starting address of the block is supplied.
    - Burst length is programmable as 1, 2, 4, 8 or a “full page” (entire row) with a burst terminate option.
  - Special commands are used for initialization (burst options etc.)
  - A burst operation takes ≈ 4 + n cycles (for n words)
READ burst (with auto precharge)

WRITE burst (with auto precharge)

Simplified DRAM timing (burst mode)

Simplified SDRAM Timing (burst mode)
**Error Detection in Memory**

- Add extra data bit to ensure total number of 1's is EVEN – even parity
- Parity bit is the exclusive-OR of all other data bits
- If one of the received bit is wrong, then number of 1's will be odd (including parity bit) and error is detected

<table>
<thead>
<tr>
<th>Data</th>
<th>Parity bit</th>
<th>number of 1's</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1 1 0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>0 1 0 1 1 0 1</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

**Example**

- If D2 is in error, check bits P1 & P2 will be wrong:

**More on check bits**

- N check bits are sufficient for $2^N - N - 1$ data bits
  - Each data bit must affect a different combination of check bits
  - There are $2^N$ possible combinations for N check bits
  - Of these, N affect only 1 bit and 1 affects no bits at all
- If you regard the combinations as binary numbers, then for 3 check bits, numbers 0, 1, 2 and 4 are ruled out, leaving 3, 5, 6, 7 as useful:

<table>
<thead>
<tr>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>P0</td>
<td>P1</td>
<td>P2</td>
<td></td>
</tr>
</tbody>
</table>

**Error Correction in Memory**

- Simply parity is limiting in its ability to detect error
- It cannot CORRECT error
- Use multiple check bits, each giving parity for a DIFFERENT COMBINATION of data bits
- If a data bit is wrong, it will affect only some check bits – from which deduce WHICH bit is wrong:

<table>
<thead>
<tr>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0:</td>
<td>D0 ⊕ D1 ⊕ D3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1:</td>
<td>D0 ⊕ D2 ⊕ D3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2:</td>
<td>D1 ⊕ D2 ⊕ D3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Code:** 3 5 6 7 (0, 1, 2, 4 not used)
ECC for 8 bit bytes

- Detect and correct any single bit error
- Can add an extra check bit giving overall parity to both data & check bits
  - will detect double bit errors but not correct
  - For double errors, overall parity bit will be OK, but check bits is wrong

<table>
<thead>
<tr>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D6</th>
<th>D7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

F0:  x  x  x  x  x
F1:  x  x  x  x  x
F2:  x  x  x  x  x
F3:  x  x  x  x  x

Code: 3 5 6 7 9 10 11 12 (0, 1, 2, 4, 8 not used, 13, 14, 15 spare)

Error detection in block of data

- Method 1: Add up all data words, ignore carry(s) and store result as CHECKSUM
  - Advantages: Very easy to do
  - Disadvantages: Errors can cancel out; will not detect gross errors such as MSB stuck at 0

- Method 2: Form checksum as before, but add an extra 1 each time you get a carry.
  - Equivalent to doing addition modulo 2^n-1 instead of module 2^n, where n is the number of bits in each data word
  - Advantages: Almost as easy as before; less likely to ignore gross errors
  - Disadvantages: Errors can still cancel – e.g. a ‘1’ changes to ‘0’ and later a ‘0’ changes to ‘1’ in the same bit

Error detection in block of data - CRC

- Method 3: Cyclic Redundancy Check (CRC)
  - Send data through a shift register incorporating a feedback loop
  - Implement ‘polynomial division’ check because the action of the shift register is mathematically equivalent to dividing one algebraic polynomial by another polynomial.
  - Shown here is one using polynomial x^16+x^12+x^5+1 : used by floppy disks
  - Whatever is left in shift register after all data bits are received is used as a checksum

![CLOCK](image)

![DATA](image)

Primitive Polynomial Table

<table>
<thead>
<tr>
<th>Degree (n)</th>
<th>Polynomial</th>
<th>Degree (n)</th>
<th>Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, 3, 4, 6, 7, 15, 22, 60, 63</td>
<td>x^n + x + 1</td>
<td>12</td>
<td>x^n + x^n + x^n + x^n + 1</td>
</tr>
<tr>
<td>5, 11, 21, 29, 35</td>
<td>x^n + x + 1</td>
<td>33</td>
<td>x^n + x^n + 1</td>
</tr>
<tr>
<td>8, 19, 38, 43</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>34</td>
<td>x^n + x^n + x^n + x^n + 1</td>
</tr>
<tr>
<td>9, 39</td>
<td>x^n + x^n + 1</td>
<td>36</td>
<td>x^n + x^n + 1</td>
</tr>
<tr>
<td>10, 17, 25, 28, 31, 41, 52</td>
<td>x^n + x^n + 1</td>
<td>37</td>
<td>x^n + x^n + x^n + x^n + 1</td>
</tr>
<tr>
<td>13, 24, 45, 64</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>40</td>
<td>x^n + x^n + x^n + x^n + 1</td>
</tr>
<tr>
<td>14, 16</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>42</td>
<td>x^n + x^n + x^n + x^n + 1</td>
</tr>
<tr>
<td>18, 57</td>
<td>x^n + x^n + 1</td>
<td>46</td>
<td>x^n + x^n + x^n + x^n + 1</td>
</tr>
<tr>
<td>23, 47</td>
<td>x^n + x^n + 1</td>
<td>54</td>
<td>x^n + x^n + x^n + x^n + 1</td>
</tr>
<tr>
<td>20, 87</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>55</td>
<td>x^n + x^n + 1</td>
</tr>
<tr>
<td>50, 51, 53, 61, 70</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>58</td>
<td>x^n + x^n + 1</td>
</tr>
<tr>
<td>32, 48</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>65</td>
<td>x^n + x^n + 1</td>
</tr>
<tr>
<td>44, 50</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>69</td>
<td>x^n + x^n + x^n + x^n + 1</td>
</tr>
<tr>
<td>49, 68</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>71</td>
<td>x^n + x^n + 1</td>
</tr>
<tr>
<td>56, 59</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>72</td>
<td>x^n + x^n + x^n + x^n + 1</td>
</tr>
<tr>
<td>66, 74</td>
<td>x^n + x^n + x^n + x^n + 1</td>
<td>73</td>
<td>x^n + x^n + 1</td>
</tr>
</tbody>
</table>