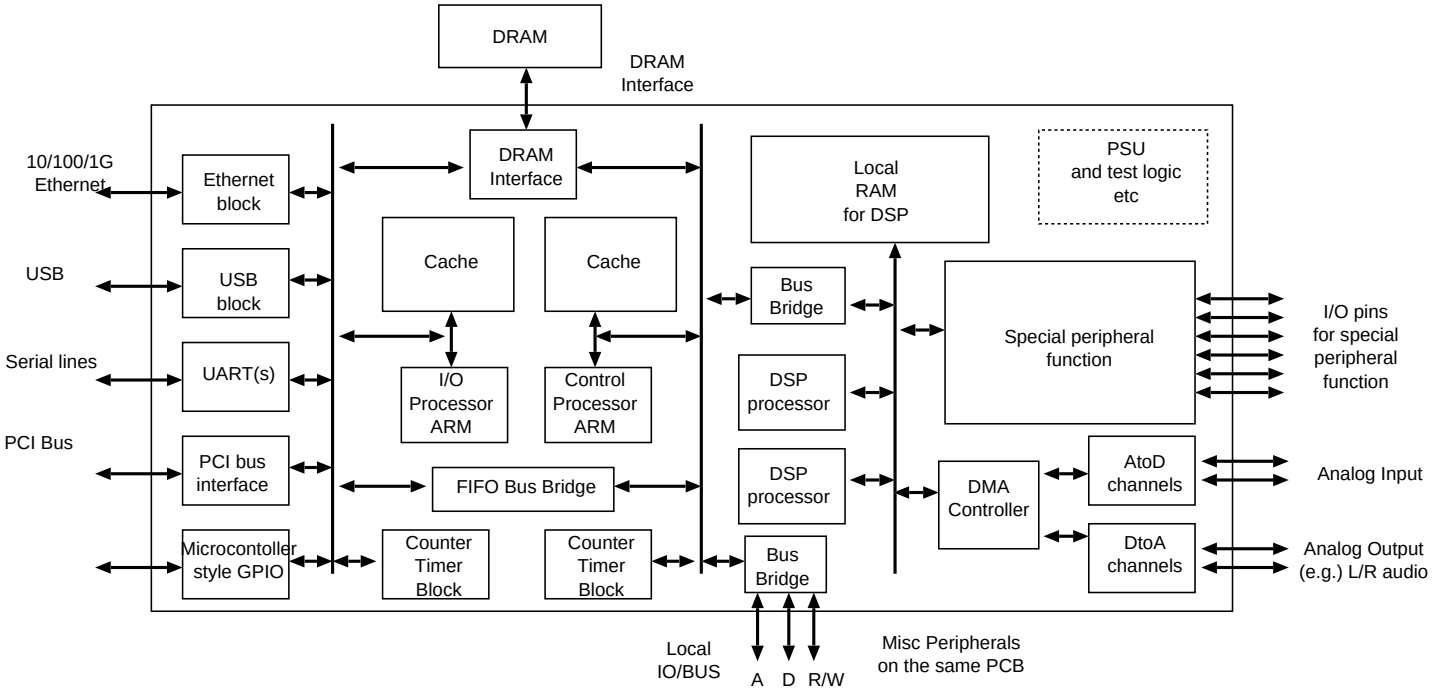
LG3: SoC System Design (SD)

Topics: System Design, SoC Tour: Typical Structure and Components.

- LG3.1 - SD System On Chip: System Design
- LG3.2 - SD Microcomputer Processor Bus
- LG3.3 - SD Microcontroller
- LG3.4 - SD Switch/LED I/O
- LG3.5 - SD Programmed I/O
- LG3.6 - SD Interrupt Structure
- LG3.7 - SD GPIO
- LG3.8 - SD Keyboard Controller
- LG3.9 - SD Counter/Timer
- LG3.10 - SD Video Controller
- LG3.11 - SD DMA Controller
- LG3.12 - SD Network Device
- LG3.13 - SD Bus Bridge
- LG3.14 - SD Clock Tree
- LG3.15 - SD Clock Domain Crossing

LG3.1 - SD: System On Chip: System Design

System on a Chip = SoC design.



Platform Chip: Example Virata Helium 210
`www.icfinder.co.kr/pdfview.asp?id=2965`.

LG1.1 - SD: Platform Chip Description

Our platform chip has two ARM processors and two DSP processors. Each ARM has a local cache and both store their programs and data in the same offchip DRAM.

The left-hand-side ARM is used as an I/O processor and so is connected to a variety of standard peripherals. In any typical application, many of the peripherals will be unused and so held in a power down mode.

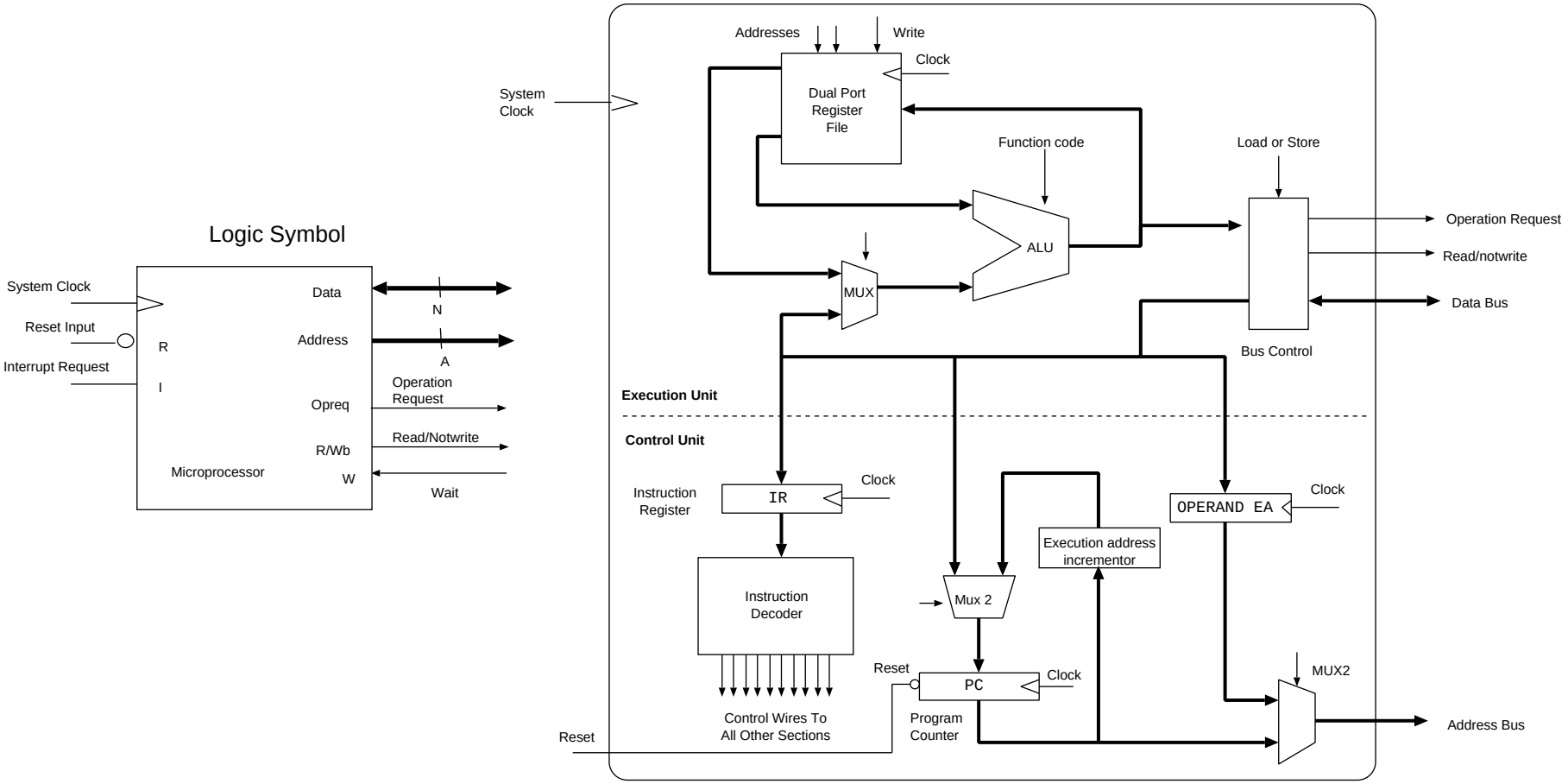
The right-hand-side ARM is used as the system controller. It can access all of the chip's resources over various bus bridges. It can access off-chip devices, such as an LCD display or keyboard via a general purpose A/D local bus.

The bus bridges map part of one processor's memory map into that of another so that cycles can be executed in the other's space, allbeit with some delay and loss of performance. A FIFO bus bridge contains its own transaction queue of read or write operations awaiting completion.

The twin DSP devices run completely out of on-chip SRAM. Such SRAM may dominate the die area of the chip. If both are fetching instructions from the same port of the same RAM, then they had better be executing the same program in lock-step or else have some own local cache to avoid huge loss of performance in bus contention.

The rest of the system is normally swept up onto the same piece of silicon and this is denoted with the 'special function periperhal.' This would be the one part of the design that varies from product to product. The same core set of components would be used for all sorts of different products, from iPODs, digital cameras or ADSL modems.

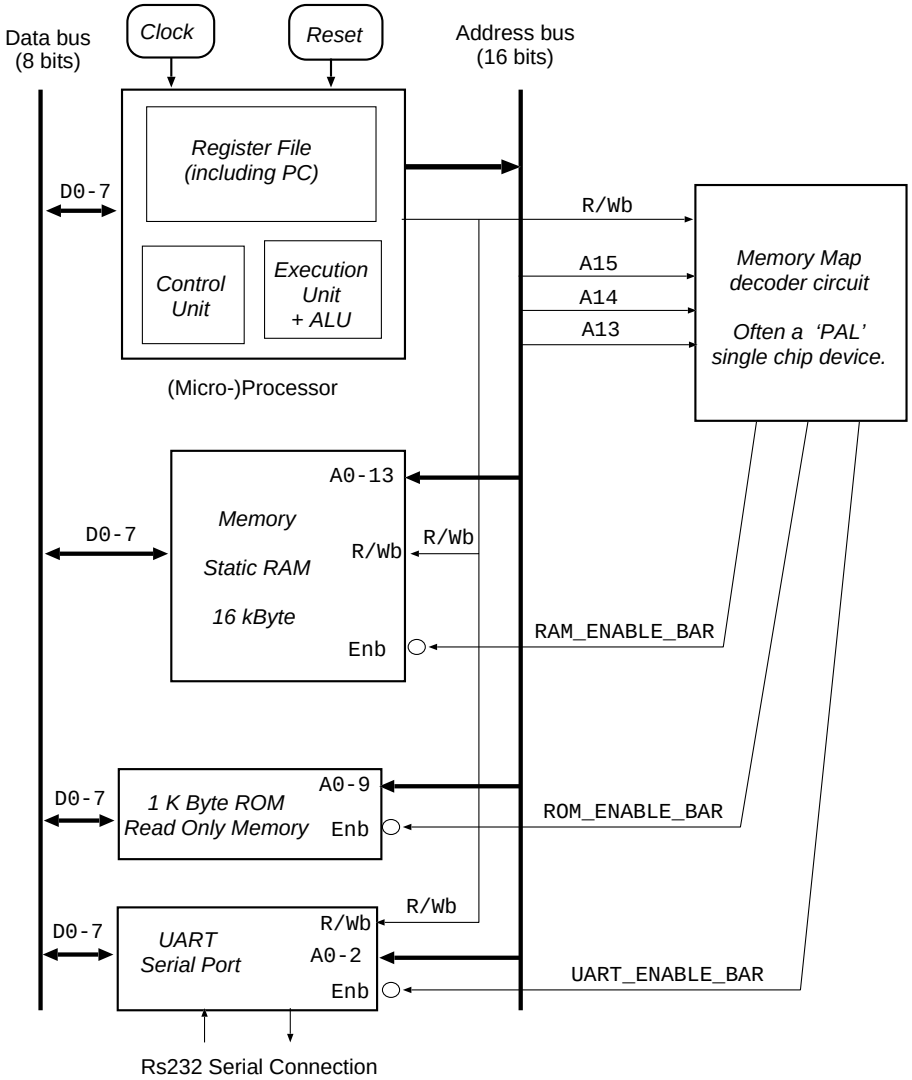
LG3.2 - SD: Microcomputer: Processor Bus Structure



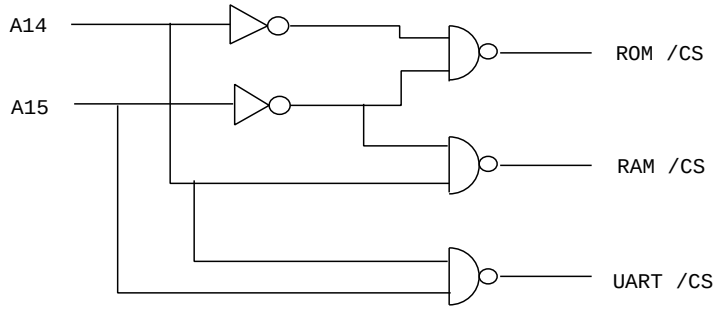
A microprocessor logic symbol and simplified internal structure.

This device is a bus master: or initiator in ESL terms.

LG3.2c - SD: A D8/A16 Computer



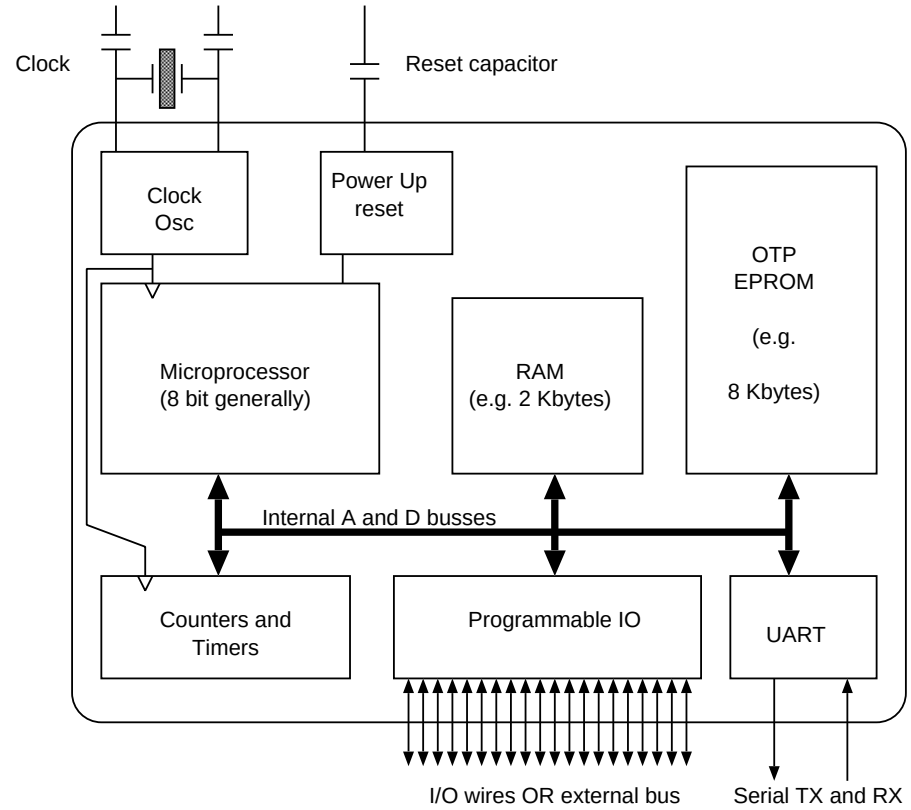
LG3.2c -SD: Memory Address Mapping and Decode



Start	End	Resource
0000	03FF	EPROM
0400	3FFF	Unused images of EPROM
4000	7FFF	RAM
8000	BFFF	Unused
C000	C001	Registers in the UART
C002	FFFF	Unused images of the UART

```
module address_decode(abus, rom_cs, ram_cs, uart_cs);
   input [15:14] abus;
   output rom_cs, ram_cs, uart_cs;

   assign rom_cs = (abus == 2'b00);  // 0x0000
   assign ram_cs = (abus == 2'b01);  // 0x4000
   assign uart_cs = !(abus == 2'b11);// 0xC000
endmodule
```

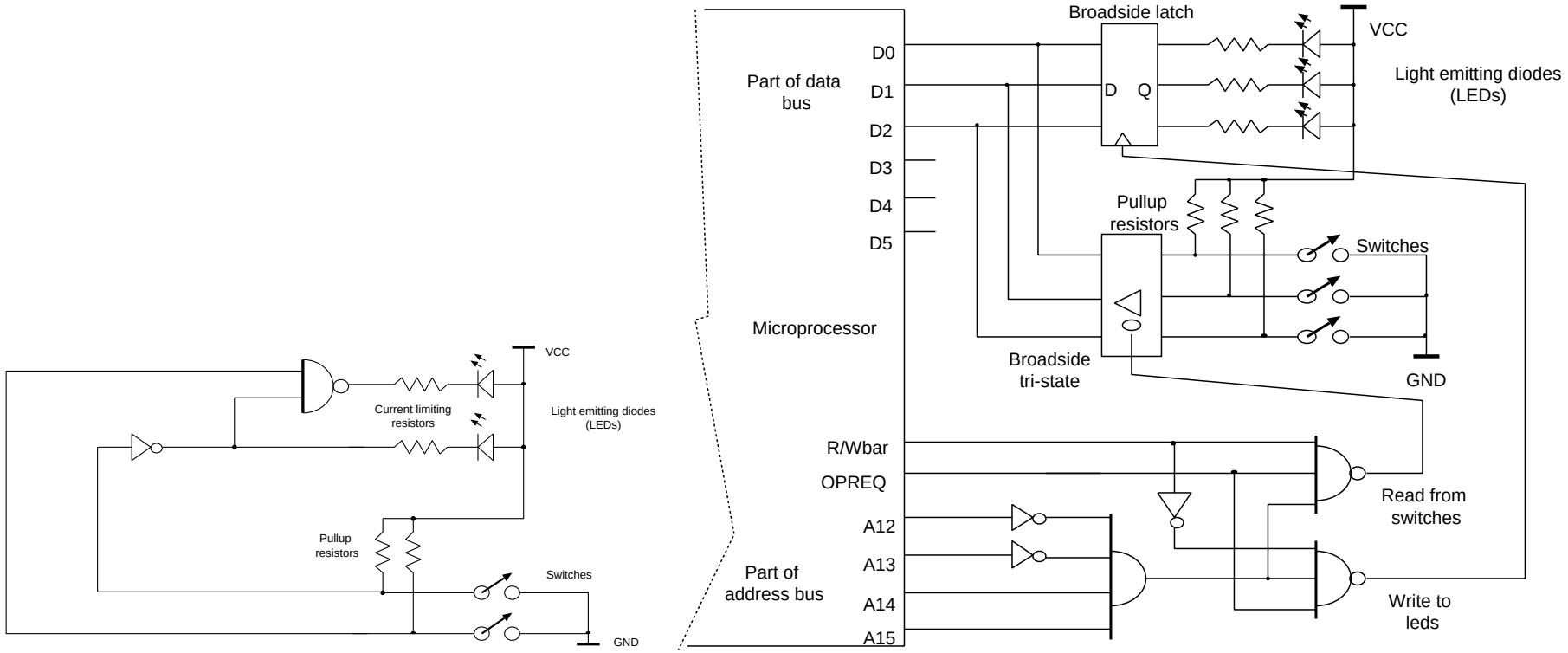
LG3.3 - SD: : A Basic Micro-Controller



Introduced 1989-85.

Such a micro-controller has an D8/A16 architecture and would be used in a mouse or smartcard.

LG3.4 - SD: Switch/LED Interfacing



a) Example of electronic wiring for switches and LEDs.

b) Example of memory address decode and simple LED and switch interfacing for programmed I/O (PIO) to a microprocessor.

LG3.5 - SD: Programmed I/O

Input and output operations where a program on a processor makes read or write operations to the device.

Inefficient - too much polling for general use.

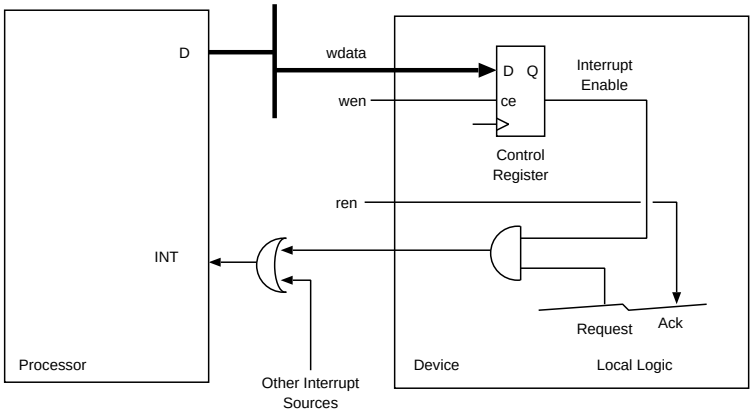
Interrupt driven I/O much better.

```
#define IO_BASE 0xFFFC1000 // or whatever

#define U_SEND    0x10
#define U_RECEIVE 0x14
#define U_CONTROL 0x18
#define U_STATUS  0x1C

#define  UART_SEND()  \
   (*((volatile char *)(IO_BASE+U_SEND)))
#define  UART_RECEIVE() \
  (*((volatile char *)(IO_BASE+U_RECEIVE)))
#define  UART_CONTROL() \
  (*((volatile char *)(IO_BASE+U_CONTROL)))
#define  UART_STATUS() \
  (*((volatile char *)(IO_BASE+U_STATUS)))

#define UART_STATUS_RX_EMPTY (0x80)
#define UART_STATUS_TX_EMPTY (0x40)

#define UART_CONTROL_RX_INT_ENABLE (0x20)
#define UART_CONTROL_TX_INT_ENABLE (0x10)
```

```
char uart_polled_read()
{
   while (UART_STATUS() &
      UART_STATUS_RX_EMPTY) continue;
   return UART_RECEIVE();
}
```

```
uart_polled_write(char d)
{
   while (!(UART_STATUS() &
     UART_STATUS_TX_EMPTY)) continue;
   UART_SEND() = d;
}
```

LG3.6 - SD: Interrupt Structure

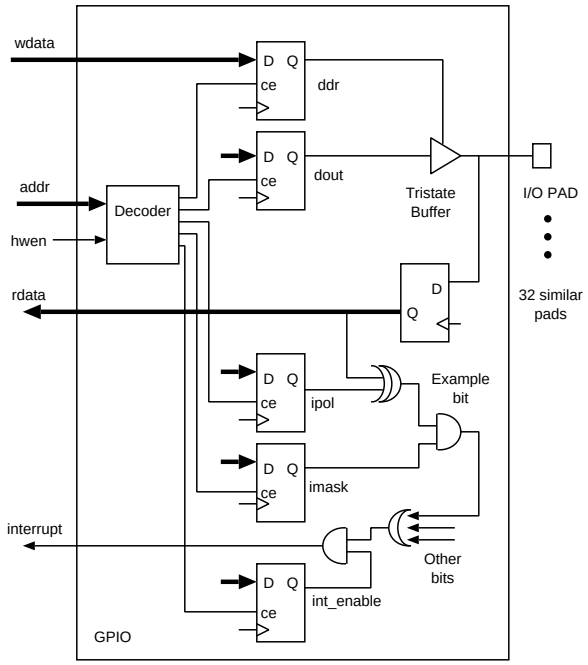


Receiving device: Keep int enabled: dev interrupts when data ready.

Transmit device: Enable int when s/w output queue non-empty: dev interrupts when h/w output queue has space.

Enchancement: Vectored interrupt tells processor which device and priority.

LG3.7 - SD: GPIO - General Purpose Input/Output Pins

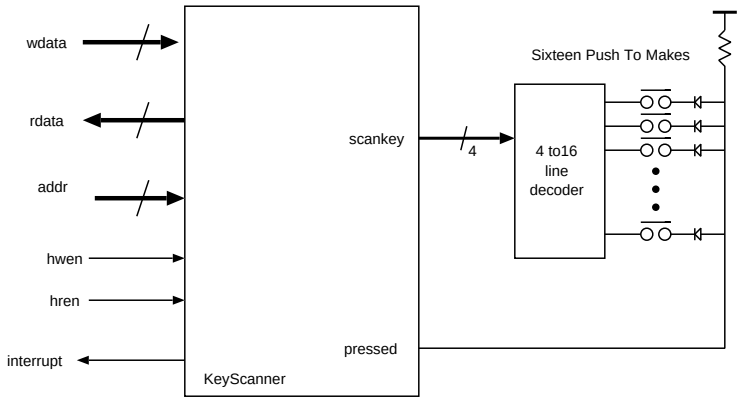



```
reg [31:0] ddr;    // Data direction reg
reg [31:0] pins_r; // regsiter'd pin data
reg [31:0] dout;   // output regsiter
reg [31:0] imask;  // interrupt mask
reg [31:0] ipol;   // interrupt polarities

reg int_enable, // output register

always @(posedge clk) begin
   pins_r <= pins;
   if (hwen && addr==0) ddr <= wdata;
   if (hwen && addr==4) dout <= wdata;
   if (hwen && addr==8) imask <= wdata;
   if (hwen && addr==12) ipol <= wdata;
   if (hwen && addr==16) int_enable <= wdata[0];
   end

bufif b0  (pins[0], dout[0], ddr[0]); // Tri-state buf
.. // thirty others here
bufif b31 (pins[31], dout[31], ddr[31]);

wire int_pending = (|((din ^ ipol) & imask));
assign rdata = pins_r;
assign interrupt = int_pending && int_enable;
```

Micro-controllers have a large number of GPIO. Platform chips have a few.

Ex: Show how to wire up a push button and write a device driver that counts pressed.

LG3.9 - SD: Keyboard Scanning Controller




```
output [3:0] scankey;
input pressed;
reg int_enable, pending;
reg [3:0] scankey, pkey;

always @(posedge clk) begin
   if (!pressed) pkey <= scankey;
   else scankey <= scankey + 1;

   if (hwen) int_enable <= wdata[0]
   pressed1 <= pressed;
   if (!pressed1 && pressed) pending <= 1;
   if (hren) pending <= 0;
   end
assign interrupt = pending && int_enable;
assign rdata = { 28'b0, pkey };
```

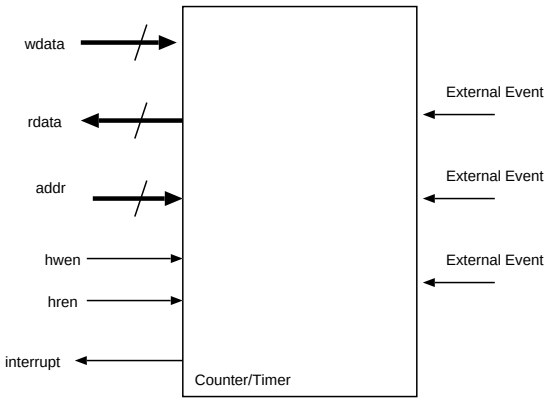
In practice, scan more slowly and use extra register on asynchronous input `pressed`.

Could use a separate microcontroller to scan keyboard.

This keyboard scanner generates an interrupt on each key press.

Standard PC keyboard generates an output byte on press and release and implements a short FIFO.

LG3.8 - SD: Counters and Timers




```
reg [31:0] prescale, prescalar;
reg [31:0] counter, reload;
reg int_enable, ovf, int_pending;

always @(posedge clk) begin
   ovf <= (prescale == prescalar);
   prescale <= (ovf) ? 0: prescale+1;
   if (ovf) counter <= counter -1;
   if (counter == 0) begin
       int_pending <= 1;
       counter <= reload;
       end
   if (host_op) int_pending <= 0;
   end
wire host_op = hwen && addr == 32;
assign interrupt = int_pending && int_enable;
```

Re-load register accommodates poor interrupt latency.

Timer (illustrated) : counts pre-scaled system clock.

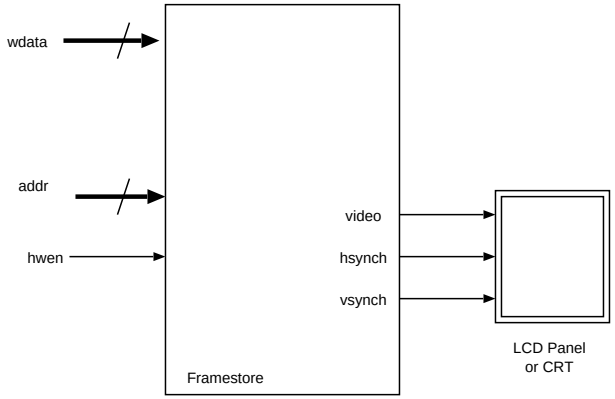
Counter: counts external input pulses (e.g. car rev counter).

Four to eight, versatile, configurable counter/timers provided in one block.

All registers also configured as bus slave read/write resources.

Interrupt cleared by host programmed I/O to `host_op`.

LG3.10 - SD: Video Controller: Framestore



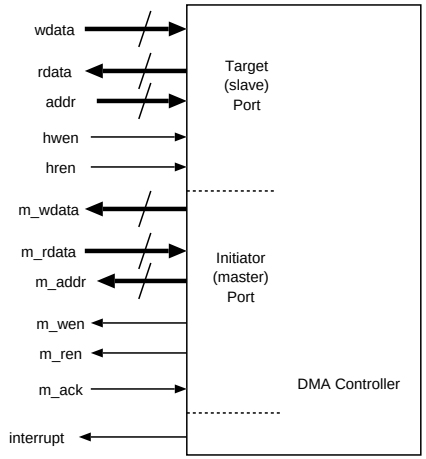

```
reg [3:0] framestore[32767:0];
reg [7:0] hptr, vptr;
output reg [3:0] video;
output reg hsynch, vsynch;

always @(posedge clk) begin
     hptr <= (hsynch) ? 0: hptr + 1;
     hsynch <= (hptr == 230)
     vptr <= (vsynch) ? 0: vptr + 1;
     vsynch <= (vptr == 110)
     video <= framestore[{vptr[6:0], hptr}];

     if (hwen) framestore[haddr]<= wdata[3:0];
     end
```

Uses private SRAM instead of main system RAM (share instead with staging FIFO?).

The pixel clock rate, H/W dimensions and synch pulse widths are normally programmable.

The RAM cannot be read. Moreover it has two address ports: re-code with one arbitrated port ?

LG3.11 - SD: DMA Controller



This controller just block copies: may need to keep src and/or dest constant for device access.

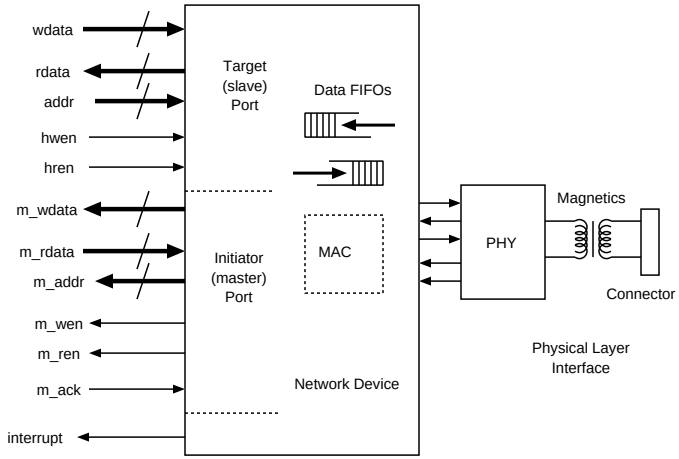
Typically, a multi-channel DMA controller is provided.

Or just use another (simple) processor ?

DMA controllers may be built into devices: SoC bus master ports needed.

```
reg [31:0] count, src, dest, datareg;
reg int_enable, active, intt, rwbar;

always @(posedge clk) begin
   if (hwen && addr==0) begin
       { int_enable, active } <= wdata[1:0];
       int  <= 0; rwbar <= 1;
       end

   if (hwen && addr==4) count <= wdata;
   if (hwen && addr==8) src <= wdata;
   if (hwen && addr==12) dest <= wdata;

   if (active && rwbar && m_ack) begin
         datareg <= m_rdata;
         rwbar <= 0;
         src <= src + 4;
         end
   if (active && !rwbar && m_ack) begin
         rwbar <= 1;
         dest <= dest + 4;
         count <= count - 1;
         end
   if (count==1 && active && !rwbar) begin
         active <= 0;
         intt <= 1;
         end
   end
assign m_wdata = datareg;
assign m_ren = active && rwbar;
assign m_wen = active && !rwbar;
assign m_addr = (rwbar) ? src:dest;
assign interrupt = intt && int_enable;
```

LG3.12 - SD: Network Device



Example: Ethernet, USB, Firewire, 802.11.

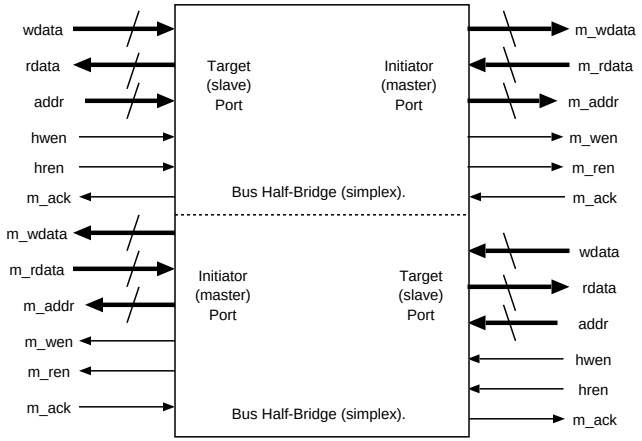
For high throughput should likely be bus master or use a DMA channel.

More importantly: DMA requires less staging RAM or FIFO in device.

Shared RAM pool: statistical multiplexing gain.

The device driver will set up a circular buffer or linked list of buffers.

LG3.13 - SD: Bus Bridge



Cycles slaved on one side are mastered on the other.

Need not be symmetric, or have flat address space.

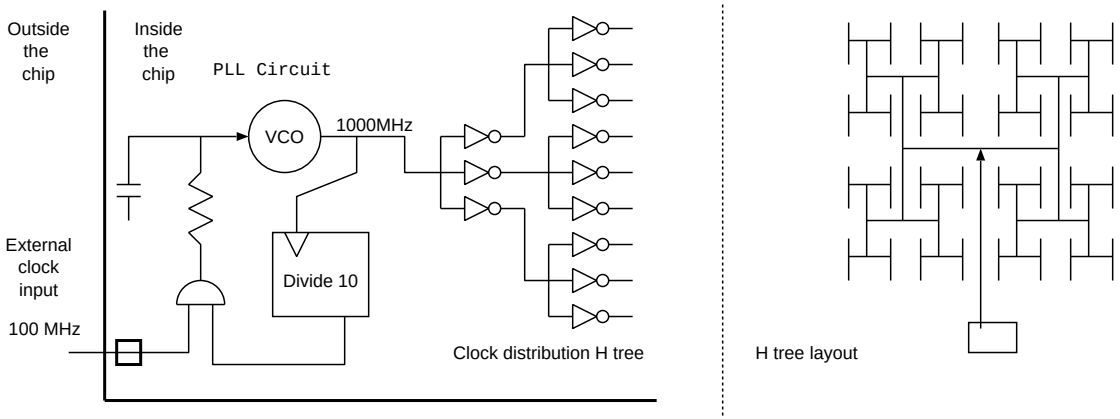
Busses can be dissimilar.

Writes posted (internal FIFO).

(The ‘busses’ on each side use multiplexors and not tri-states.)

System bandwidth ranges from 1.0 to 2.0 bus bandwidth: inverse proportion to bridge crossing cycles.

LG3.14 - SD: Clock Tree



Clock sourced from a lower-frequency external (quartz) reference.

Multiplied up internally with a phase-locked loop.

Dynamic frequency scaling (future topic): programmable PLL ratio.

Skew in delivery is minimised using a balanced clock distribution tree.

Physical layout: fractal of H's, ensuring equal wire lengths.

Inverters are used to minimise pulse shrinkage (duty-cycle distortion).

LG3.15 - SD: Clock Domain Crossing

Like a bus bridge, but different clocks on each side.

- Have one signal that is a guard or qualifier signal for all the others going in that direction.
- Make sure all the other signals are settled in advance of guard.
- Pass the guard signal through two registers before using it (metastability).
- Use a wide bus (crossing operations less frequent).

```
input clk;  // receiving domain clock

input [31..0] data;
input req;
output reg ack;

reg [31:0] captured_data;
reg r1, r2;
always @(posedge clk) begin
     r1 <= req;
     r2 <= r1;
     ack <= r2;
     if (r2 && !ack) captured_data <= data;
```

Simplex: can never be sure about the precise delay.

Need protocol with insertable/deletable padding symbols that have no semantic meaning.

100 percent utilisation impossible.

The four-phase handshake limits utilisation to 50 % (or 25 if registered at both sides)

Duplex: cannot reply on any precise timing relationship between the two directions. Protocol must rely on sequencing or expicit transaction tokens.