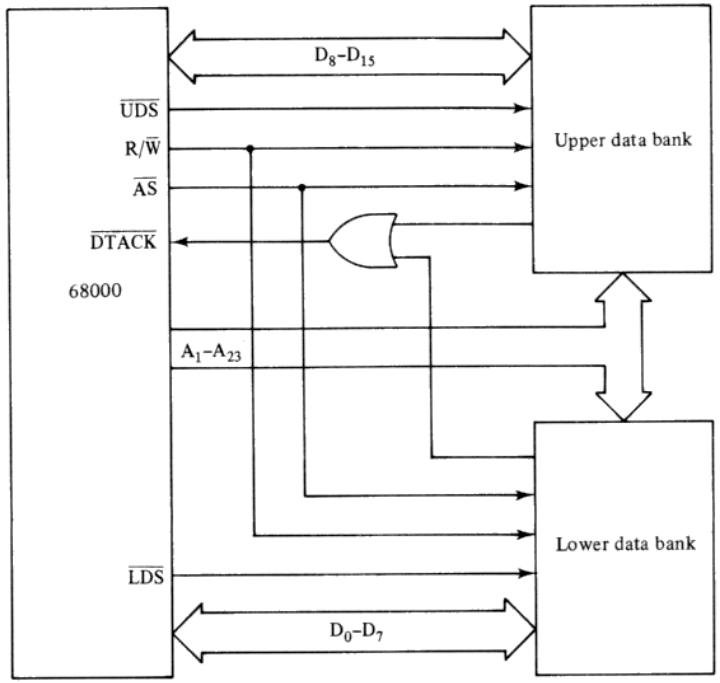
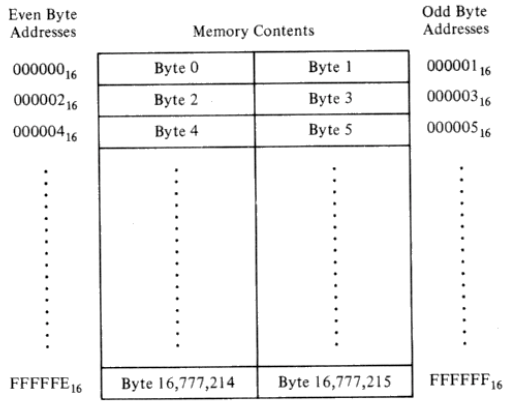
Lecture 15: Memory and I/O interface

- **Address space**
- **Memory organization**
- **Asynchronous data transfers**
  - Read and Write cycles
  - DTACK* generation
- **Synchronous data transfers**
- **Direct Memory Access**
- **System control signals**

Address space

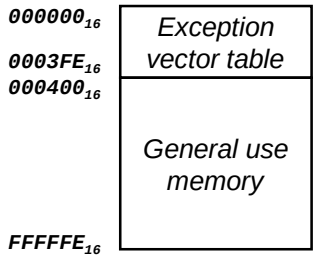
- **The address bus of the 68000 consists of 23 independent address lines A0-A23**
  - Since the data bus is 16-bit wide, we have an address space of 8M words
- **Two additional control lines are used to select individual bytes**
  - Upper Data Strobe (UDS*)
  - Lower Data Strobe (LDS*)
- **UDS* and LDS* are used to enable byte-wide *upper* and *lower* data banks**
  - Upper data banks are selected by UDS* and contain data bits D_8-D_{15}
  - Lower data banks are selected by LDS* and contain data bits D_0-D_7
- **A_{01}-A_{23} are shared by the two data banks and are used to address individual bytes within each bank**
- **This physical organization of memory explains why words must be stored at even memory addresses**



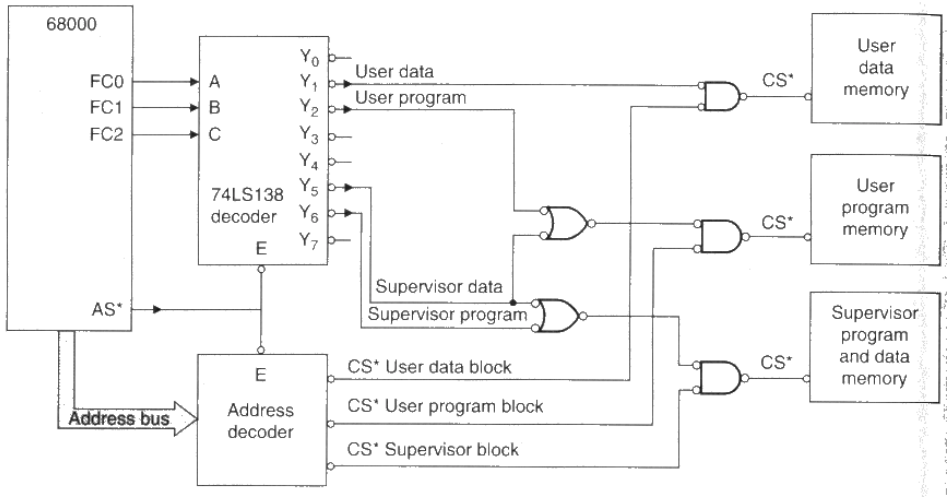


Memory organization

- **Dedicated and general use memory**
  - Memory locations 000000 to 0003FE have a dedicated function:
    - storage of the interrupt vector table
  - The rest of the memory space is for general use, it can be used to store data, instructions or address information
- **Three additional output pins on the 68000, the function code (FC_0-FC_2), indicate whether the bus cycle is**
  - accessing data or program instructions
  - being executed in user or supervisor mode
- **By using (FC_0-FC_2), the memory space can be further divided into**
  - User program segment
  - User data segment
  - Supervisor program segment
  - Supervisor data segment
- **As an example, (FC_0-FC_2) are decoded and used to assert the appropriate chip select (CS*) signals**

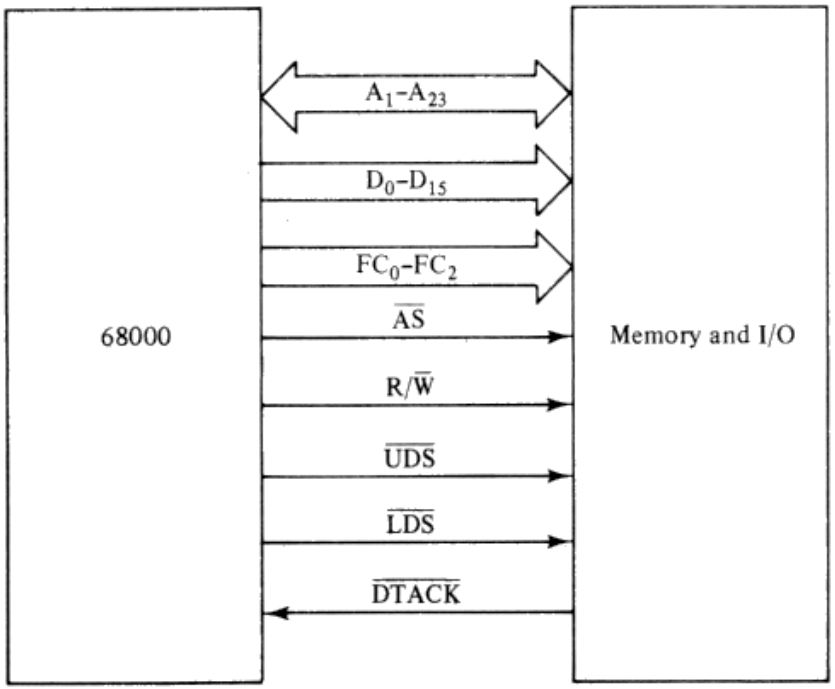


Function code			Reference class
FC_2	FC_1	FC_0	
0	0	0	(Unassigned)
0	0	1	User data
0	1	0	User program
0	1	1	(Unassigned)
1	0	0	(Unassigned)
1	0	1	Supervisor data
1	1	0	Supervisor program
1	1	1	Interrupt Acknowledge



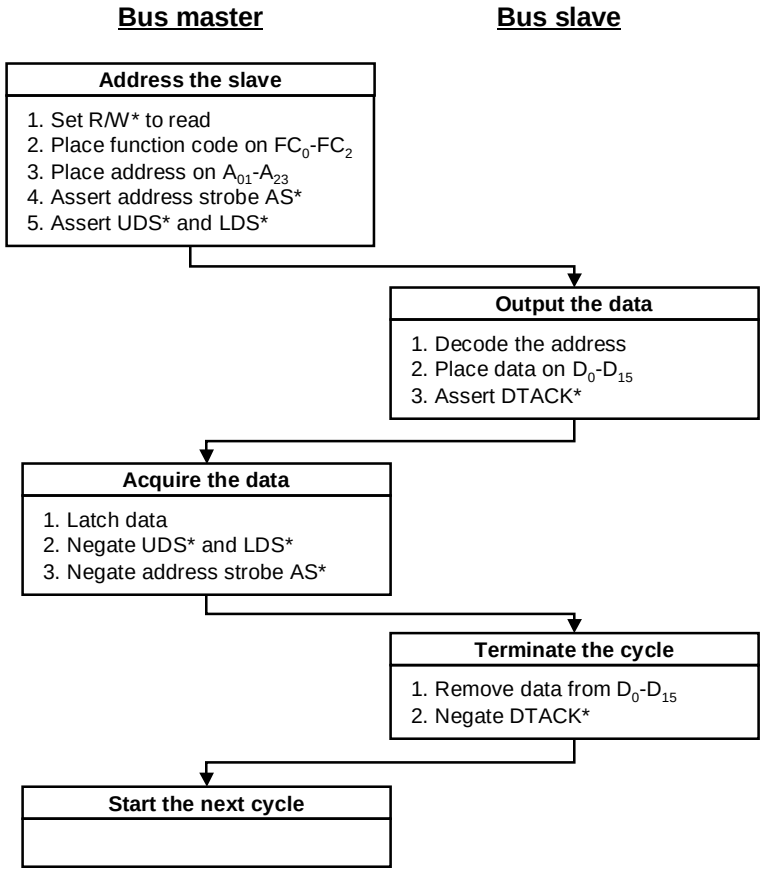
Asynchronous memory and I/O interface

- **Asynchronous means that**
  - once a bus cycle is initiated to read or write instructions or data, it is not completed until a response is provided by the memory or I/O subsystem
  - This response is an acknowledgement signal that tells the 68000 that the current bus cycle is compete
- **The basic asynchronous operation is**
  - The 68000 puts an address on the address bus and asserts Address Strobe (AS*) to signal memory and I/O devices that a valid address information is available on the bus
  - The memory or I/O device asserts Data Transfer Acknowledge (DTACK*) to signal the 68000 that
    - valid data is available on the data bus during a read operation (the 68000 latches data when DTACK* is asserted)
    - data has been successfully written to the memory or I/O device
- **Asynchronous operation allows the 68000 to interface with slow memories or I/O devices**



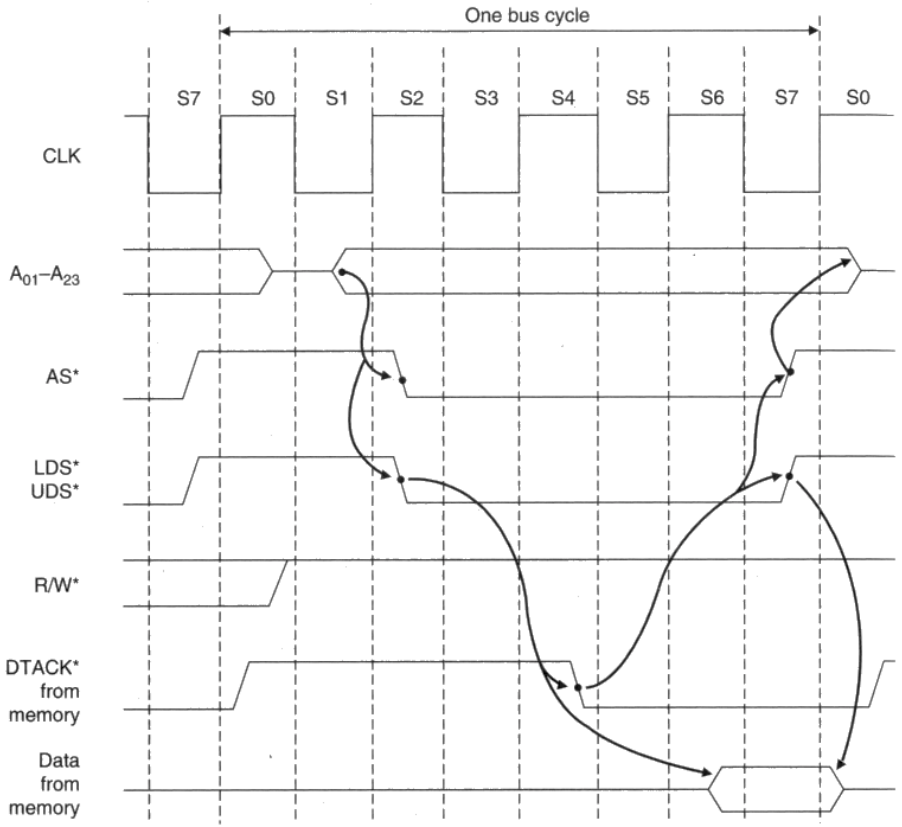
READ cycle flowchart

- **An asynchronous read cycle is characterized by the interlocked handshaking procedure that takes place between**
  - the bus master (CPU) and
  - the bus slave (memory or I/O device)
- **The following steps take place**
  - The CPU indicates its intentions by
    - forcing R/W* HIGH and
    - setting up an address
    - asserting AS*, UDS* and LDS*
  - The slave detects that AS*, UDS* and LDS* are asserted and then
    - places data on the data bus
    - asserts DTACK*
  - Upon DTACK* assertion, the CPU
    - latches data from data bus
    - Negates AS*, UDS* and LDS*
  - Upon negation of AS*, UDS* and LDS*, the slave
    - removes data from data bus
    - Negates DTACK*
  - At this point we can start a new bus cycle



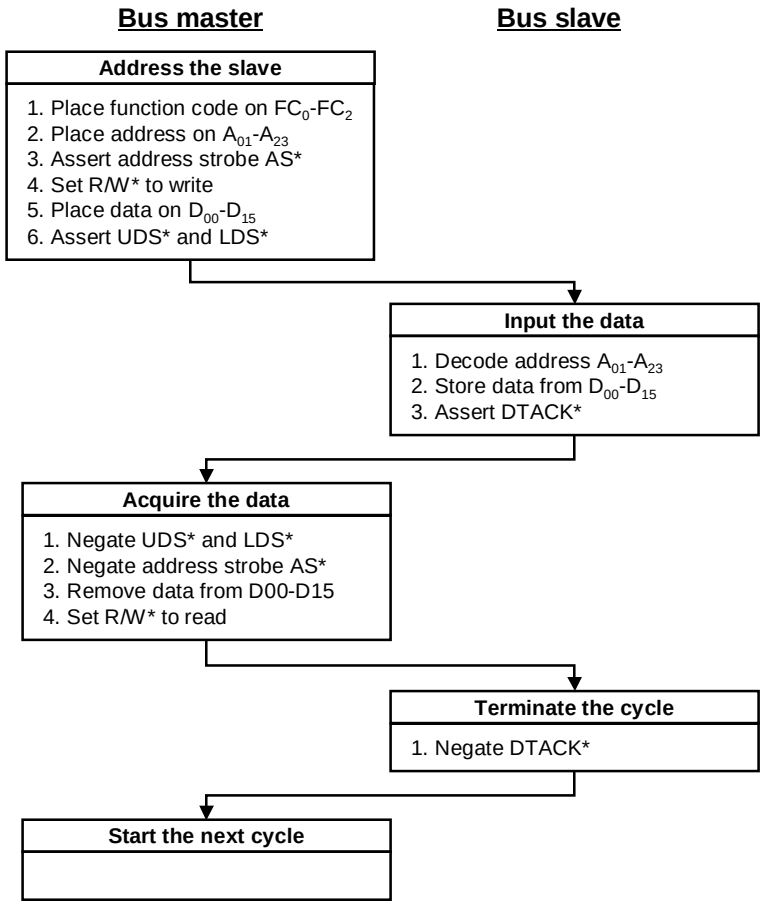
READ cycle timing diagram

- **Each bus cycle consists of a minimum of 4 clock cycles, divided into eight states S0-S7**
  - A bus cycle starts in state S0 with the clock high and ends in state S7 with the clock low
  - This basic cycle can be extended by the insertion of wait states between S4 and S5
- **Narrative**
  - During S0 all signals are negated with the exception of R/W*, which becomes high to indicate a read operation
  - In state S1, the address on A_{01}-A_{23} becomes valid and remains so until state S0 of next cycle
  - In state S2 the address strobe AS* goes active-low, indicating that the contents of the address bus are valid
  - At the same time, UDS* and LDS* go active-low and initiate the memory access
  - At the end of S4, the CPU tests DTACK*
    - If DTACK* is inactive, the CPU inserts wait states between S4 and S5 until DTACK* goes active-low on the falling edge of S4
  - During S7, the CPU
    - negates AS*, UDS* and LDS*
    - latches the data internally
  - The negation of the 3 strobes causes the memory to
    - return its data output pins to high impedance (floating) state
    - negate DTACK*



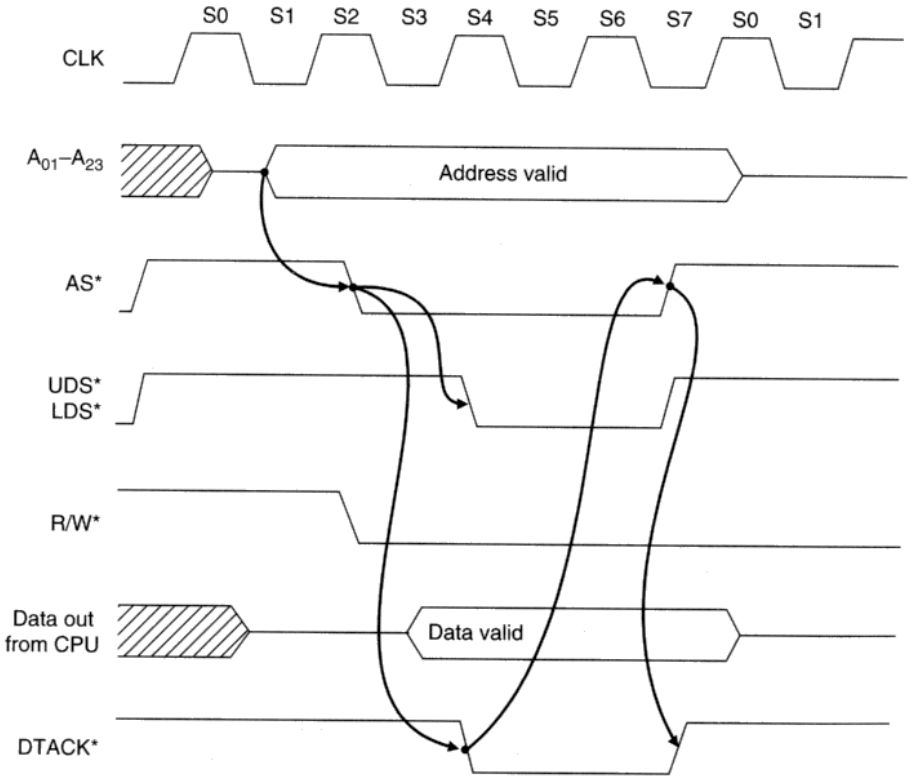
WRITE cycle flowchart

- **The asynchronous write cycle is very similar to the read cycle we saw before**
- **There are two differences**
  - The CPU provides data at the start of the write cycle
  - The bus slave reads this data



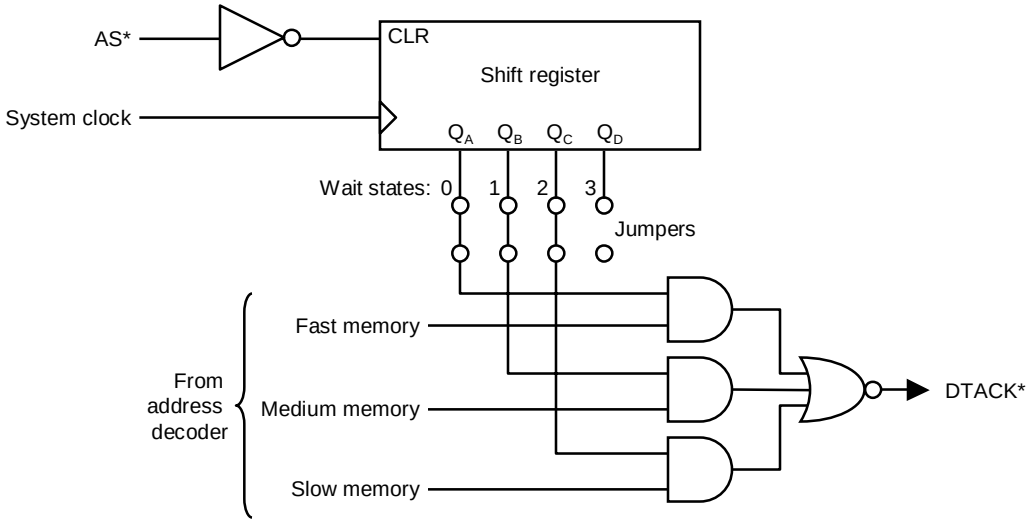
WRITE cycle timing diagram

- **Narrative**
  - At the start of a cycle, an address is placed on the address bus A_{01}-A_{23} and
    - AS* is asserted and R/W* set to logic 0
  - After R/W* has set low to indicate a write cycle, the CPU places data on the data bus
  - Once the contents of the data bus have stabilized, UDS* and LDS* are asserted (approximately 1 cycle after AS* assertion)
    - This allows the memory to use UDS* and LDS* to latch data from the bus
  - If DTACK* is asserted before the falling edge of S4, the write cycle is terminated normally
    - Otherwise wait states are inserted until DTACK* is asserted
  - At the end of the write cycle
    - AS*, UDS* and LDS* are negated simultaneously
    - R/W* is set high
    - data bus is floated



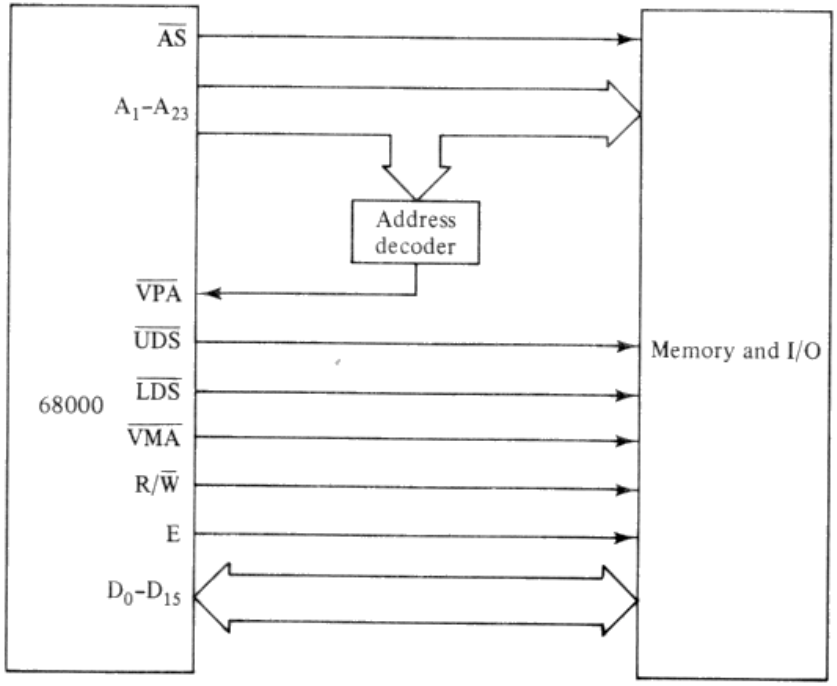
DTACK* generation

- **In the previous slides we assumed that DTACK* was generated by the memory block**
  - This is true for I/O devices such as the 68230 PI/T or the 68681 DUART, which support asynchronous data transfers and have a DTACK* output pin
  - For devices that do not have this facility (such as memory ICs), the DTACK* signal must be generated by the systems designer with additional circuitry
- **The circuit below can be used to generate the DTACK* signal with different delays depending on the speed of the device that is being accessed**
  - Q_A can be used for devices that can operate at maximum (no-wait-state) bus speed
  - Q_B can be used to delay DTACK* for one wait state (it will be set on the second rising edge of the clock)
  - Q_C can be used to delay DTACK* for two wait states, and so on...



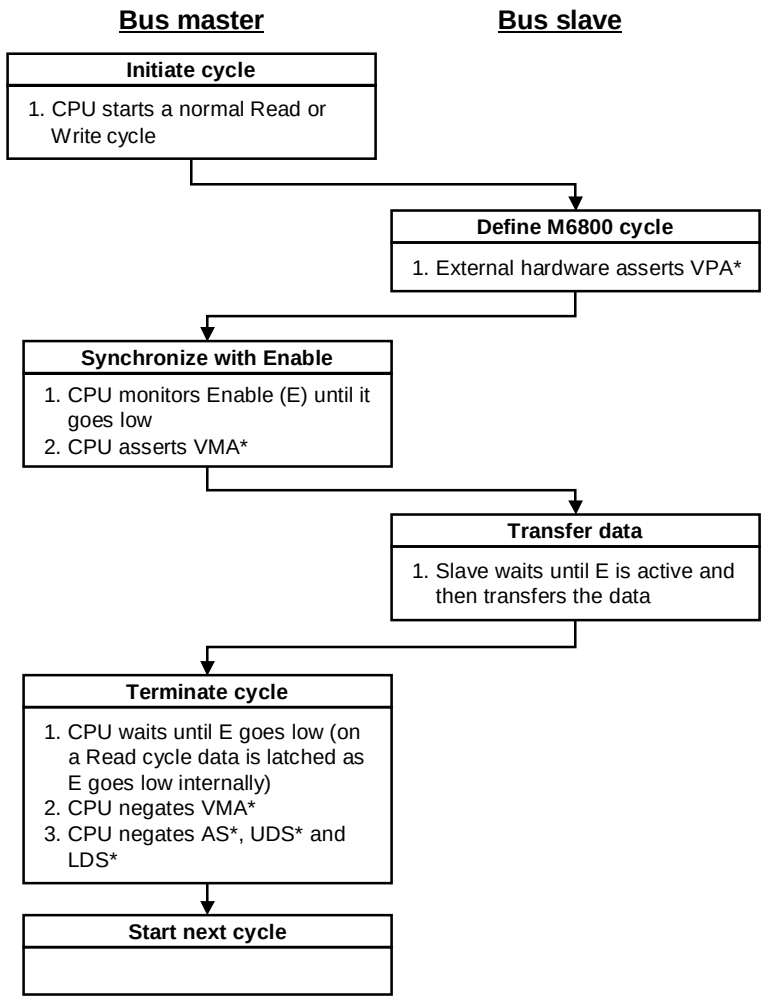
Synchronous memory and I/O interface

- **Synchronous bus operation is provided in order to interface slower 8-bit peripherals as those in the 6800 family**
- **This interface is similar to the asynchronous interface except for**
  - DTACK* is NOT used
  - Instead, three synchronous bus control signals are used:
    - Valid Peripheral Address (VPA*)
    - Valid Memory Address (VMA*)
    - Enable (E)
- **Valid Peripheral Address (VPA*)**
  - When a synchronous peripheral detects that it is being accessed, it asserts VPA* to request a synchronous bus cycle
  - When the CPU detects that VPA* has been asserted, it initiates the synchronous transfer by means of VMA* and E
- **Valid Memory Address (VMA*)**
  - The CPU asserts VMA* to indicate the peripheral that there is a valid address on the address bus
  - The assertion of VMA* is a response of the CPU to the peripheral's assertion of VPA*
- **Enable (E)**
  - A CPU output derived from the 68000 clock cycle
  - One E cycle is equal to 10 CPU cycles
  - E clock is non-symmetric it is low for 6 clock cycles and high for 4 clock cycles



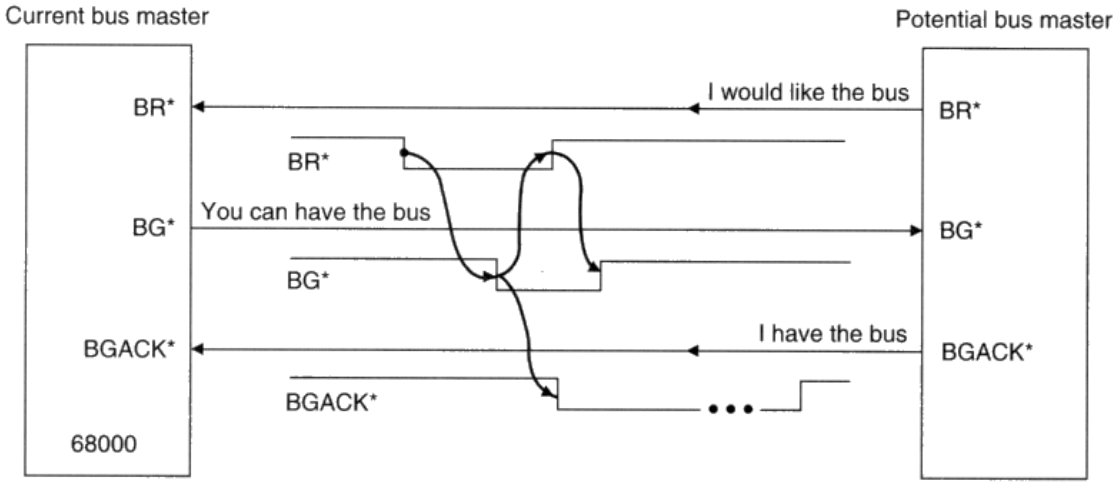
Synchronous transfer

- **Sequence of operations (not detailed)**
  - During state S0, the CPU generates the function codes FC0-FC2
  - During S1, the CPU puts a valid address on the address bus
  - When the address is stable in S2
    - AS* is asserted and R/W* set to 1 for a read or 0 for a write
    - If a write operation is performed, the data is output on D0-D15 and maintained valid for the rest of the bus cycle
  - By the end of S4, external circuitry must have decoded the address on the bus
    - At this time, it asserts VPA*
  - In response to the VPA* assertion, the CPU synchronizes with falling edge of E and then asserts VMA*
    - This signals the peripheral that a valid address is on the address bus
  - The peripheral transfers data on the next rising edge of E
    - For a read cycle the CPU latches the data on the next falling edge of E
  - At this point the data transfer is complete and the CPU negates VMA*, AS*, UDS* and LDS*



Direct Memory Access (DMA)

- **DMA is a process in which a device (slave) requests the use of the master's buses (address, data and control) for its own use**
  - In a microprocessor-based system, the master is usually the CPU
- **Once the slave device has control of the bus, it can read or write to the system memory as necessary. When the slave device is finished, it releases control of the master's buses, and the system returns to normal**
- **DMA is used to achieve faster data transfer rates than those attainable with the CPU**
- **To perform DMA on the 68000, three signals are used**
  - Bus Request (BR*)
  - Bus Grant (BG*)
  - Bus Grant Acknowledge (BGACK*)

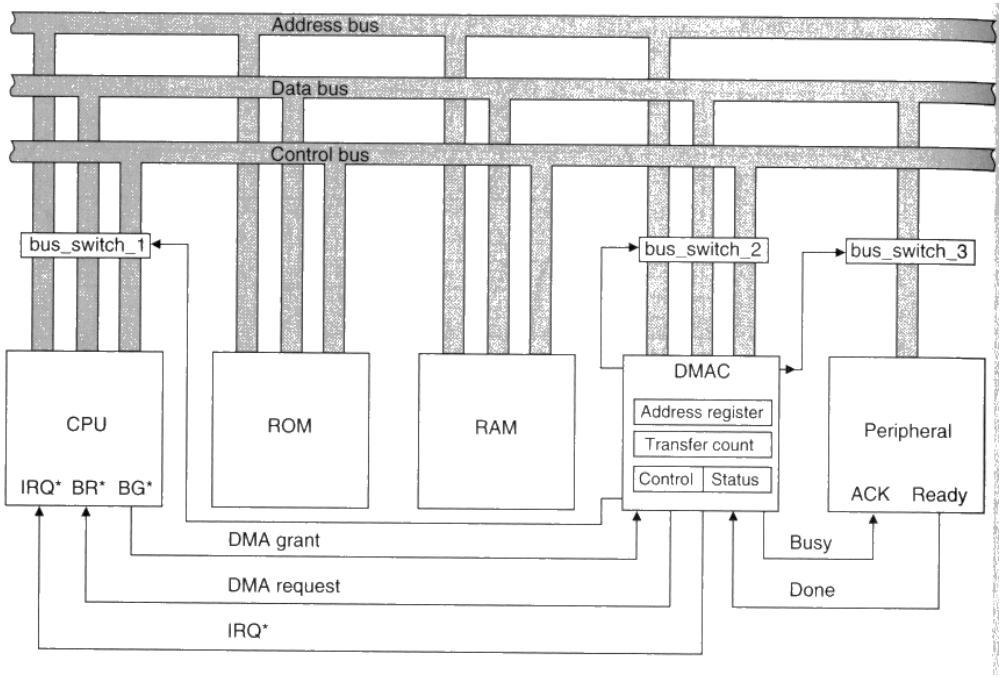


Why is DMA faster?

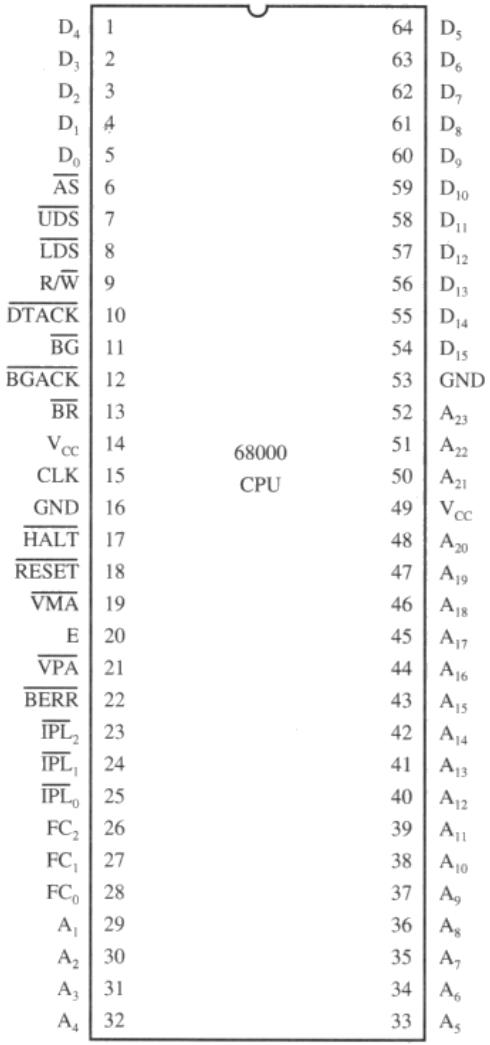
- **Consider the problem of writing a block of memory to an output port, one byte at a time. The following tasks must be performed**
  - Initialize memory and output port addresses
  - Repeat until all bytes are transferred:
    - Read byte from memory
    - Write byte to output port
    - Increment memory address
    - Check to see if all bytes transferred
    - Wait until output port ready for next byte
- **With this approach, only a fraction of the memory cycles are used for the actual data transfer**
  - the speed of the data transfer is much less than the maximum rate at which data can be read from the memory
- **DMA is performed with a device called DMA controller (DMAC), which can be thought of a very specialized microprocessor, except for**
  - unlike a data transfer performed by the CPU, no instructions need to be fetched during the transfer to tell the DMAC how to perform the transfer
  - thus, all memory cycles are available for transferring data

DMAC operation

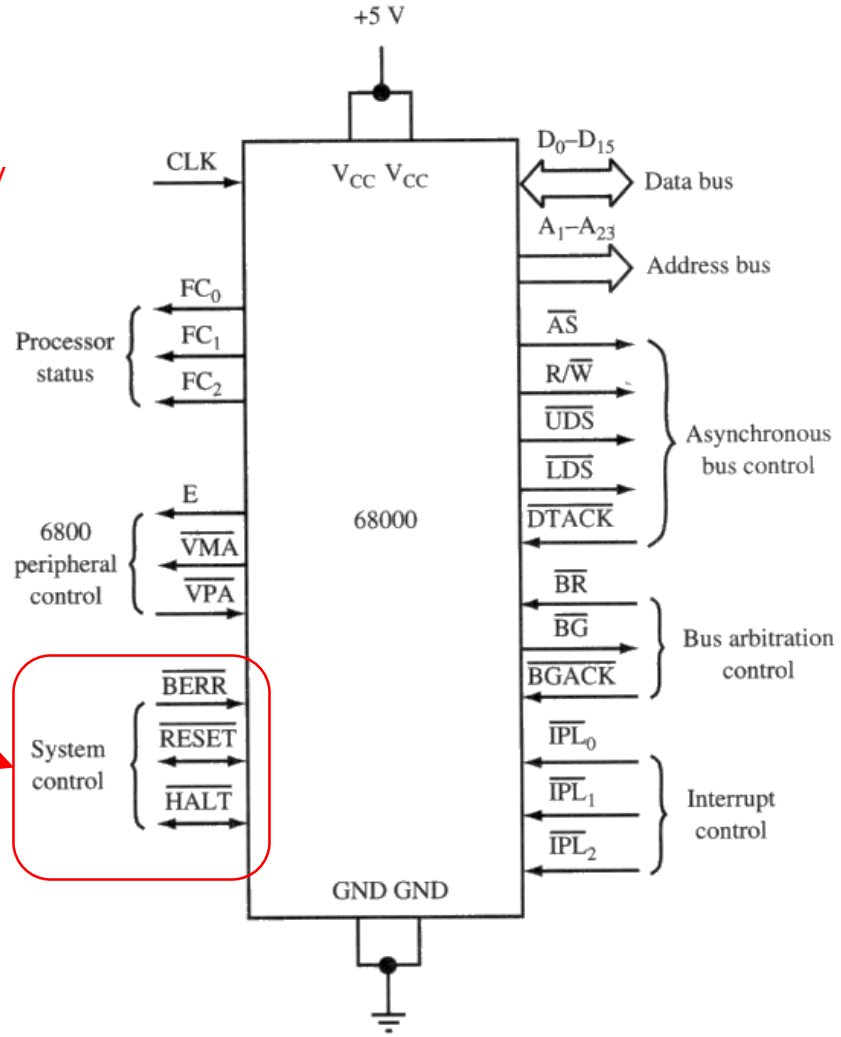
- **A DMA transfer takes place in several stages**
  - Like with any other memory-mapped peripheral, the CPU first sets up the DMAC's registers to define
    - the quantity of data to be moved
    - the type of DMA operation
    - the direction of data transfer
  - The DMAC is activated by a request for service from its associated peripheral
    - When the peripheral asserts REQ*, the DMAC requests control of the bus by
      - asserting its BR* output
      - waiting for a BG* response from the bus master
      - asserting BGACK*
  - Once the DMAC has control of the bus, it generates all timing signals needed to transfer data between peripheral and memory
  - DMA transfers take place in one of two modes
    - Burst mode: several operands are transferred in consecutive bus cycles
    - Cycle stealing mode: the DMA relinquishes the system bus between successive data transfers allowing normal CPU processing to be interleaved



MC6800 pinout



We have already studied all the pins except for these three



System control pins

- **Bus Error (BERR*)**
  - When asserted, it indicates that something has gone wrong with the current bus cycle
  - For example, an access to an invalid memory address is generated by faulty software
    - The external logic detects this error and asserts BERR* to inform the 68000
  - The action taken by the 68000 when it recognizes BERR* is rather complex and depends on the state of HALT*. For simplicity we will state that it will either
    - try to rerun the faulty cycle or
    - will generate an exception, and the OS will deal with the bus error
- **Halt (HALT*)**
  - This I/O pin can be used for three purposes
    - Used as an input: to force the 6800 to execute one cycle at a time (for debugging purposes)
    - Used as an input: to rerun a failed bus cycle (see BERR*)
    - Used as an output: when the 68000 finds itself in a situation it cannot recover from, it stops further processing and asserts HATL* to indicate this situation
- **Reset (RESET*)**
  - Used as an input: the 68000 loads the SSP from M[$000000] and PC from M[$000004]
    - The RESET* pin will be connected to the system's hardware reset button
  - Used as an output: when the processor executes the RESET command, it will assert RESET* pin to reset all external devices
    - This command does not affect the internal state of the 68000, so it allows peripherals to be reset without resetting the CPU