

# ECE 571 Group 3 Final Project Presentation

Project Advisor: Prof. Mark Faust

**RTL Design of 8237A DMA Controller for  
8086 Microprocessor based Systems**

# Team Members

- ★ Rishitosh Sawant
- ★ Poojitha Murahari
- ★ Janisha Kumar
- ★ Srinivas Rao Lakkaraju



# Agenda



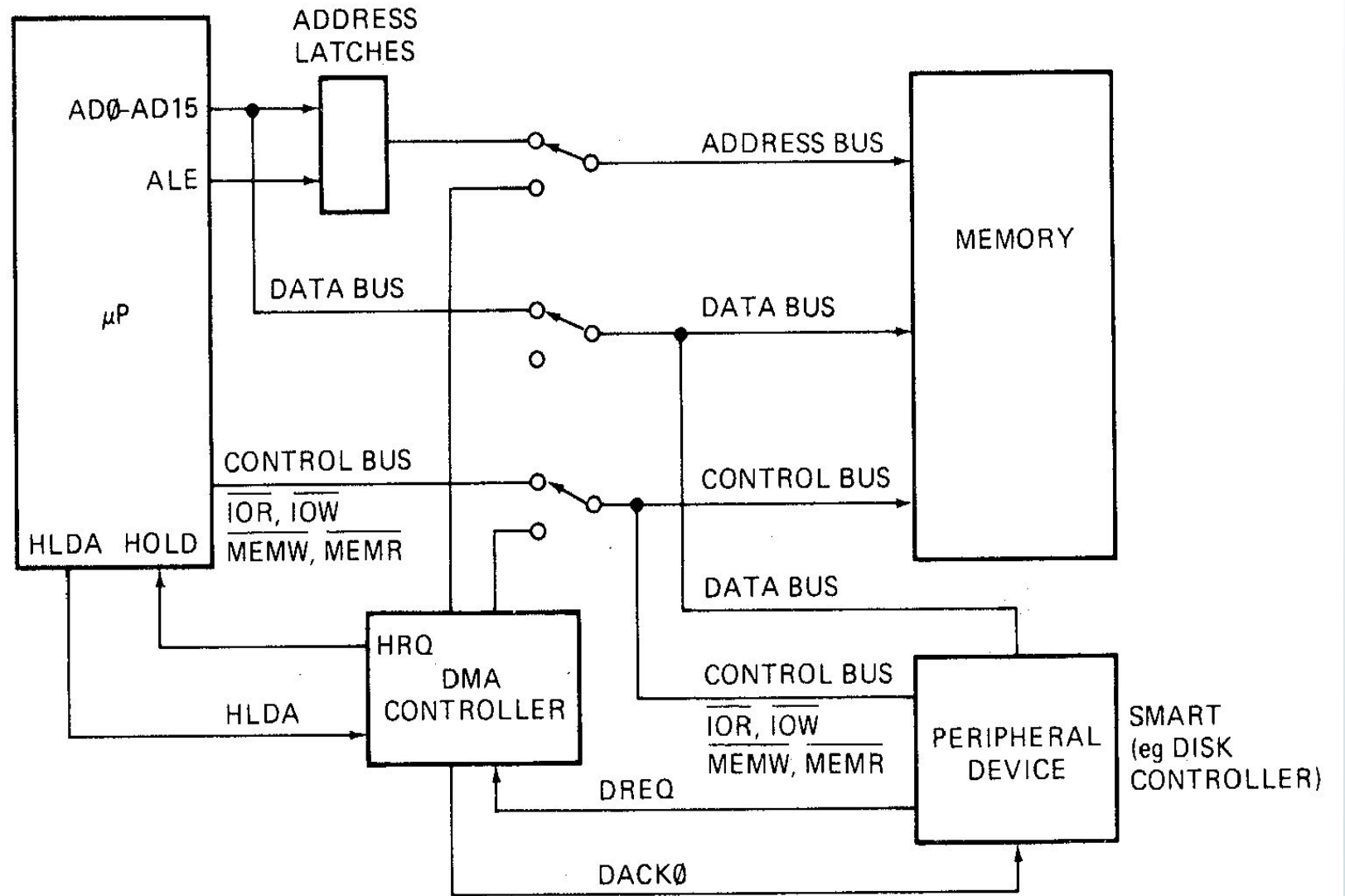
- About DMA controller
- DMA interface with 8086
- DMA design features
- Benefits of Systemverilog Interfaces
- DMA block diagram
- Bi-directional signal modeling
- System Verilog constructs used in the project
- DMA Registers
- DMA Timing command and FSM
- Timing diagram
- Priority logic
- Challenges and learnings
- Future scope
- References



# ABOUT 8237A DMA CONTROLLER

- ❑ Direct memory access (DMA) is a process in which an external device takes over the control of system bus from the CPU.
- ❑ A DMA controller interfaces with several peripherals that may request DMA.
- ❑ The controller decides the priority of simultaneous DMA requests communicates with the peripheral and the CPU, and provides memory addresses for data transfer.
- ❑ The 8237 is in fact a special-purpose microprocessor.
- ❑ Normally it appears as part of the system controller chip-sets.
- ❑ The 8237 is a 4-channel device. Each channel is dedicated to a specific peripheral device and capable of addressing 64K bytes section of memory.

## DMA interface with 8086





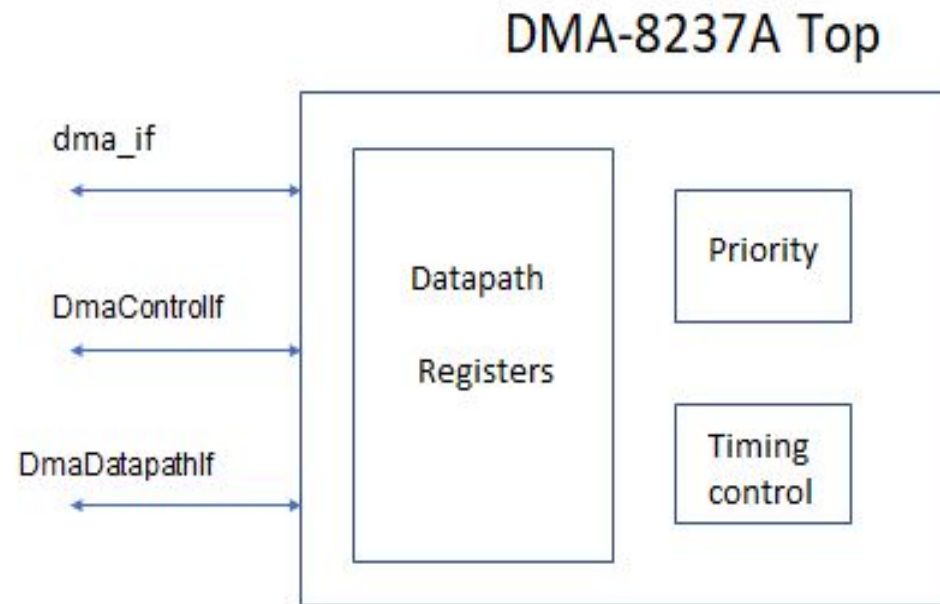
# DMA Design Features

- Rotating priority for DMA channel requests.
- Masking of channel requests on DREQ lines.
- Special SW commands for Program condition.
- Single transfer mode
- Auto Initialization of channels
- Address multiplexing to reduce pin count.
- EOP - terminating active DMA transfers.
- High performance with transfers up to 1.6MB/s running on 5Mhz.
- Four independent channels.
- Enable/disable control of individual DMA requests.
- Independent auto initialization of all channels.
- I/O to memory and memory to I/O DMA transfers.
- Address increment or decrement.
- Independent polarity control of (DREQ) Request and (DACK) Acknowledge signals.

# DMA Top

## DMA – 8237A Top

- 



## Benefits of using Systemverilog Interfaces: Top module is less than ten lines.

```
module Dma8237aTop(dma_if dif);

// DMA interface instantiation
DmaControlIf cif(dif.CLK, dif.RESET);
DmaRegIf rif(dif.CLK, dif.RESET);

// DMA modules instantiation
// Datapath module
DmaDatapath D1(dif, cif, rif);

// Timing and Control module
DmaTimingControl C1(dif, cif, rif);

// Priority logic
DmaPriority P1(dif, cif, rif);

endmodule
```



# DMA Block Diagram

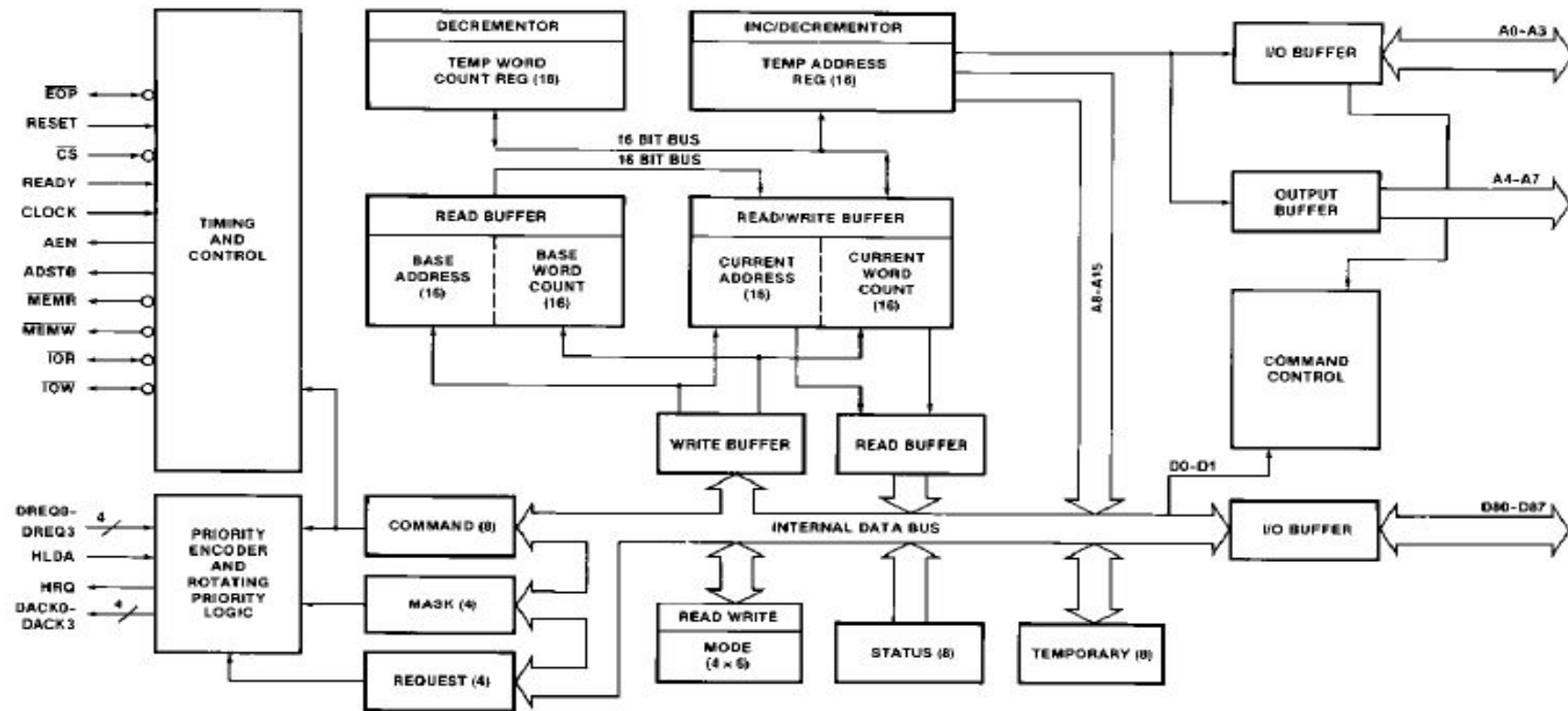


Figure 1. Block Diagram

231466-1

## Bi-directional signal modeling sample

```
// Data bus logic
always_ff@(posedge dif.CLK) if(!dif.CS_N && !dif.IOW_N) ioDataBuf <= dif.DB;
assign dif.DB = (!dif.CS_N && ~dif.IOR_N) ? ioDataBuf : 8'bz;

// Address Bus logic
always_ff@(posedge dif.CLK) if(!dif.CS_N) ioAddrBuf <= dif.ADDR_L;
assign dif.ADDR_U = (dif.CS_N) ? outAddrBuf : 4'bz;
assign dif.ADDR_L = (dif.CS_N) ? ioAddrBuf : 4'bz;
```

# Wire type for Bi-directional signals in Interface

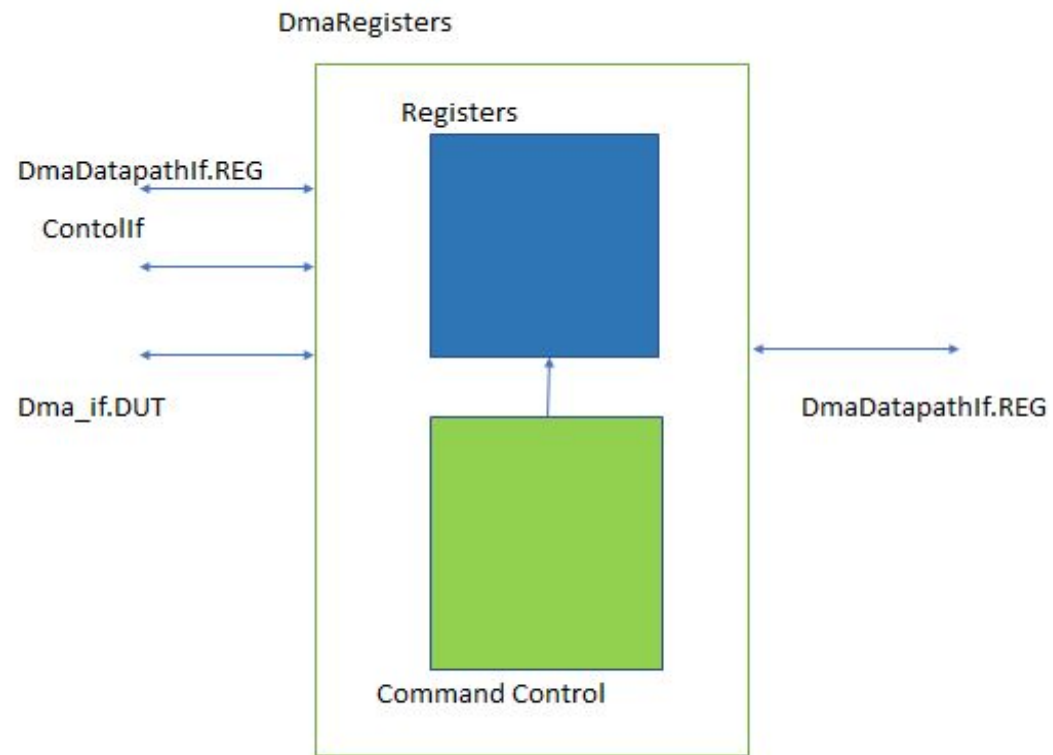
```
interface dma_if(input logic CLK, RESET);

    // interface to 8086 processor
    wire    IOR_N;           // IO read
    wire    IOW_N;          // IO write
    logic   MEMR_N;          // MEM read
    logic   MEMW_N;          // MEM write
    logic   HLDA;            // Hold acknowledge from CPU to indicate it has relinquished bus control
    logic   HRQ;             // Hold request from DMA to CPU for bus control
```

## System Verilog constructs used in the project: Priority case for priority encoder

```
// priority encoder
always_comb begin
    pencoderOut = '0; // default value when no request on DREQ lines
    if(enFixedPriority)
        priority case(1'b1) // reverse case
            cif.VALID_DREQ0 : pencoderOut = 4'b0001;
            cif.VALID_DREQ1 : pencoderOut = 4'b0010;
            cif.VALID_DREQ2 : pencoderOut = 4'b0100;
            cif.VALID_DREQ3 : pencoderOut = 4'b1000;
        endcase
    end
```

# DMA Registers



# Total number of Registers Used

Register	Num	Size(bit)
Base Address Registers	4	16
Base Count Registers	4	16
Current Address Registers	4	16
Current Count Registers	4	16
Temporary Address Register	1	8
Temporary Count Register	1	8
Mode Registers	4	6
Command Register	1	8
Request Register	1	8
Mask Register	1	8
Status Register	1	8
Temporary Register	1	8



## Register codes

Register	Operation	Signals						
		$\overline{\text{CS}}$	$\overline{\text{IOR}}$	$\overline{\text{IOW}}$	A3	A2	A1	A0
Command	Write	0	1	0	1	0	0	0
Mode	Write	0	1	0	1	0	1	1
Request	Write	0	1	0	1	0	0	1
Mask	Set/Reset	0	1	0	1	0	1	0
Mask	Write	0	1	0	1	1	1	1
Temporary	Read	0	0	1	1	1	0	1
Status	Read	0	0	1	1	0	0	0



# Software Commands

## **Clear First/Last Flip-Flop**

This command is used to set the Flip-Flop to decide the byte to be loaded into the registers.  
if the Flip-Flop is set to 1, the upper byte is loaded into the registers.  
if the Flip-Flop is set to 0, the lower byte is loaded into the registers.

## **Master Clear**

This command Clears the command, Status, Request and the other registers when required.

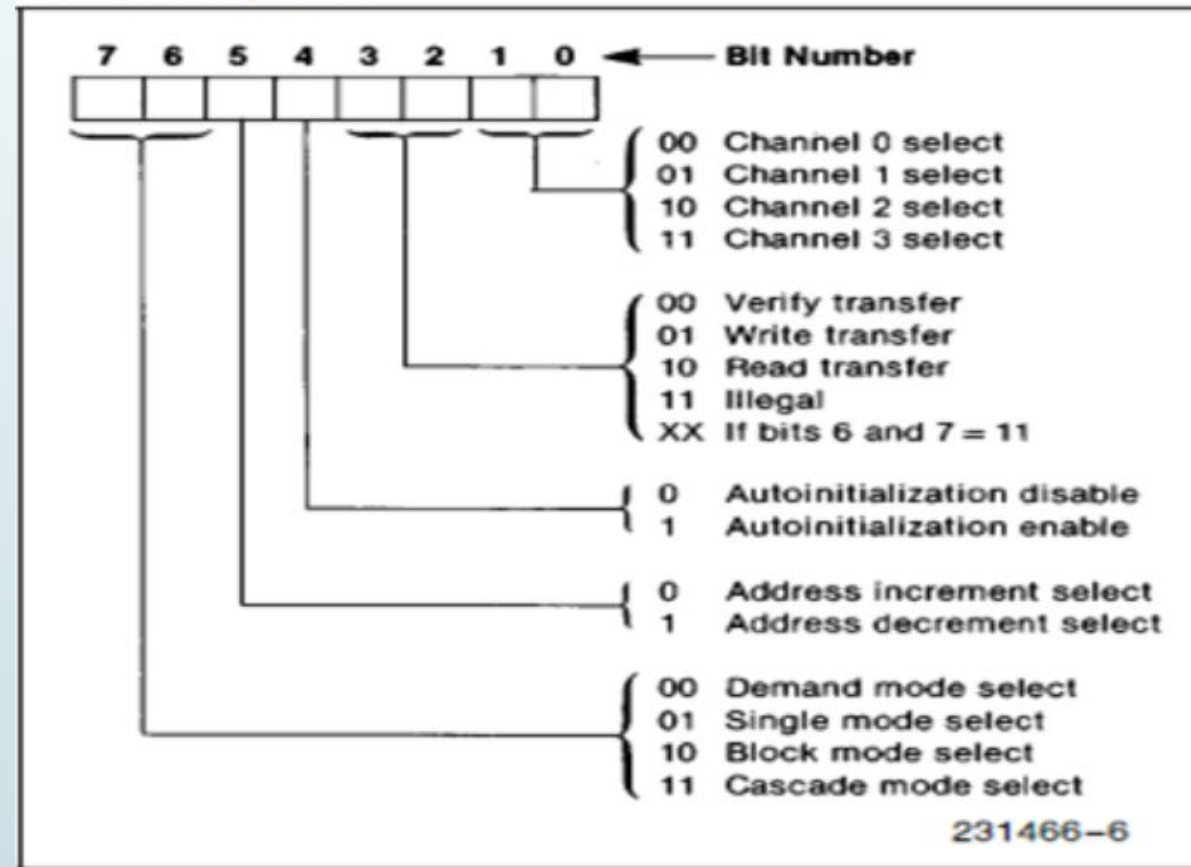
## **Clear Mask Register**

The Clear Mask Register clears the the Mask bits to enable all the channels available to the Transfer.

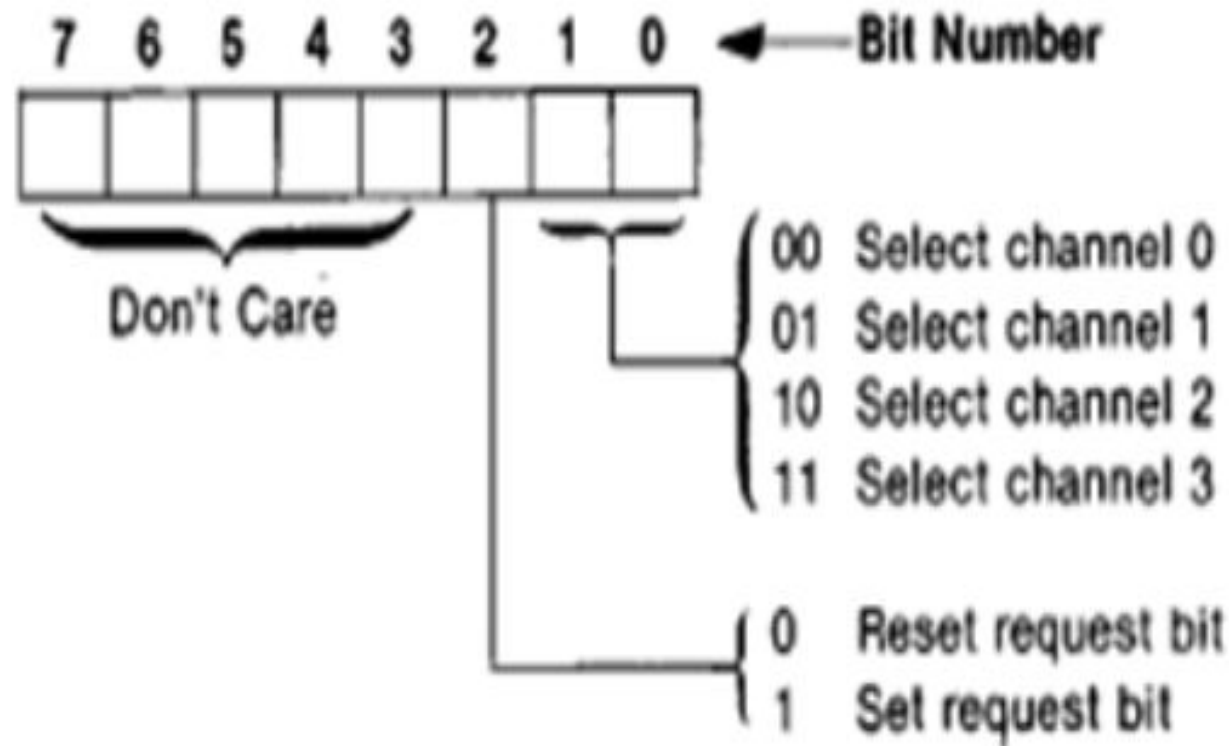


# Registers

**Mode Register**

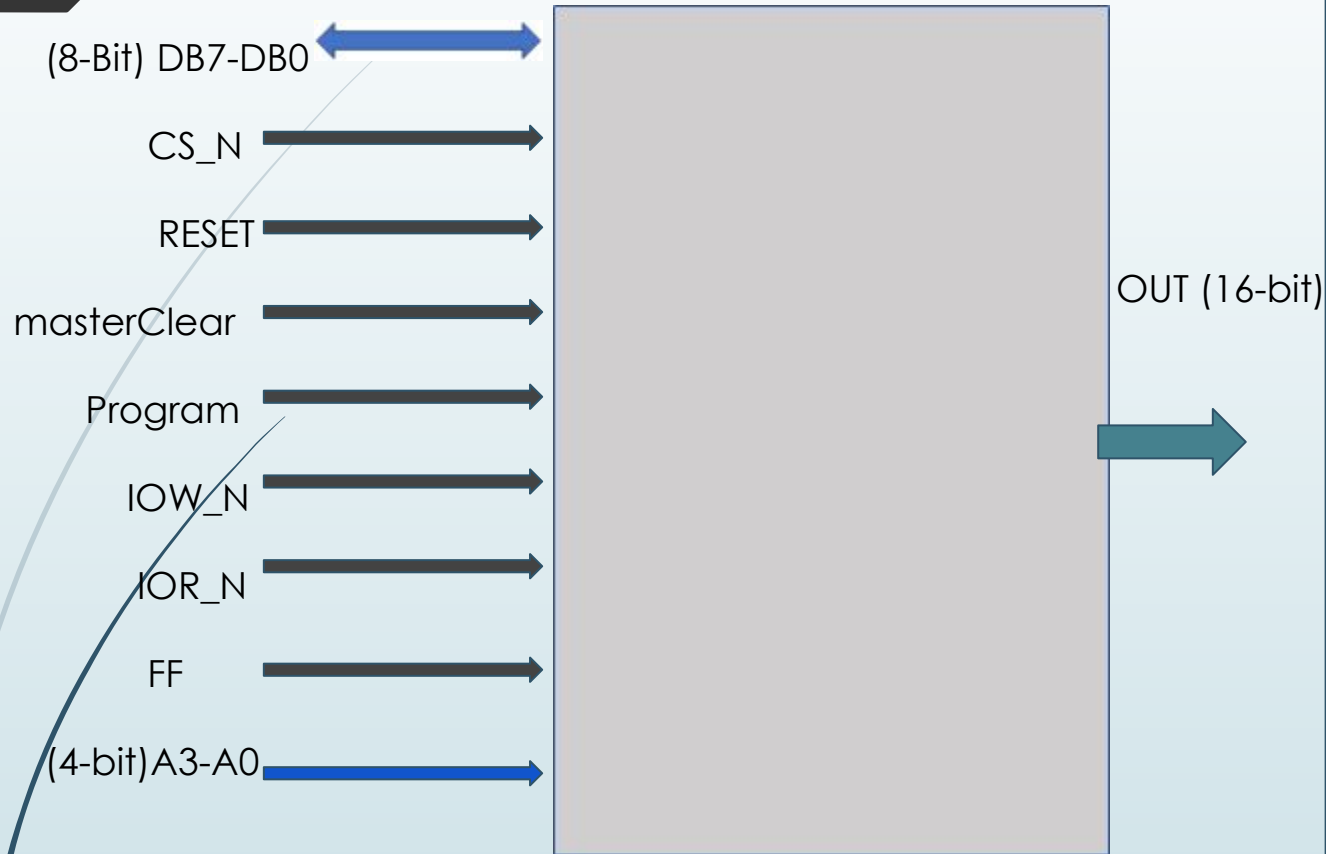


## Request Register



231466-7

## 16-Bit Registers



NOTE:  
The **ldtempCurrAddr** and **ldtempCurrWord** are internal signals to load the current Address into temporary registers.  
Program -> internal signal

The shown block diagram is the implementation of the

**Base Address Register -> 16 bit**

**Base Count Register -> 16 bit**

**Current Address Register -> 16 bit**

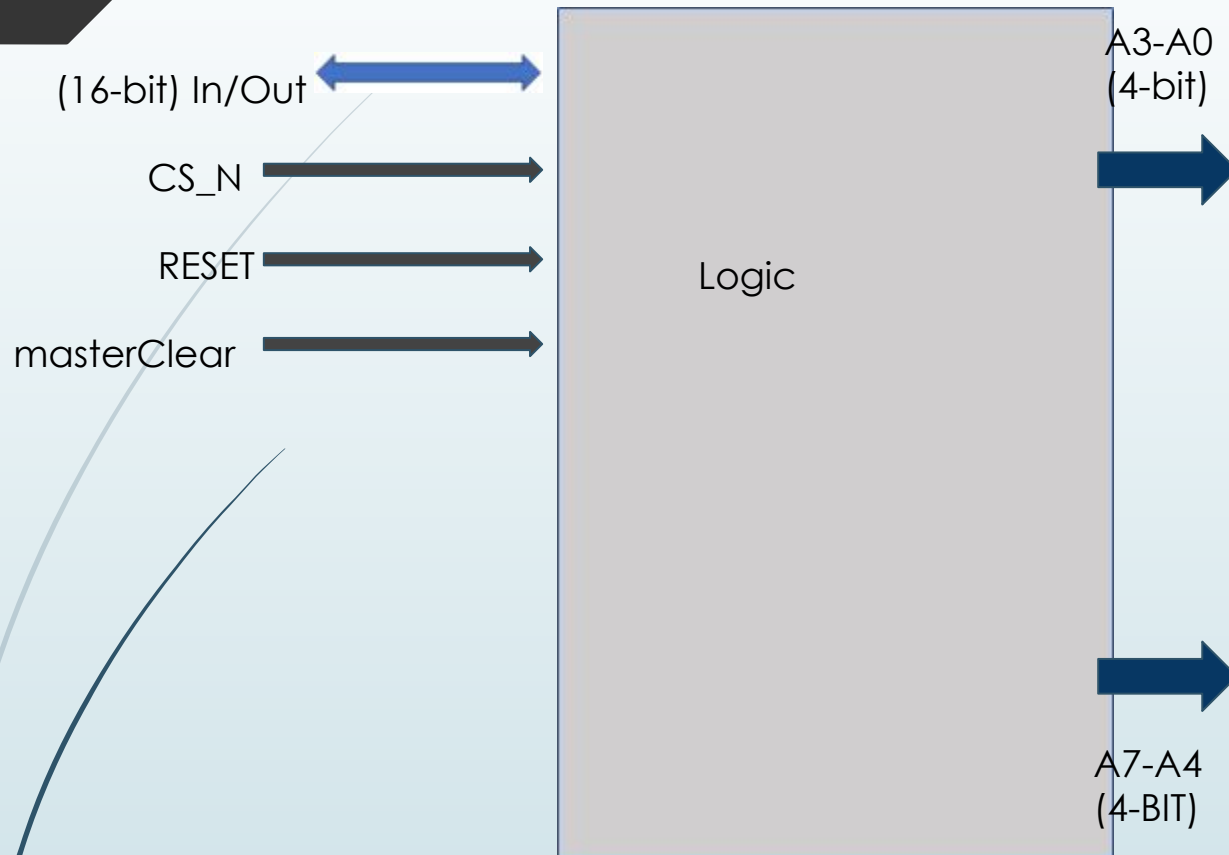
**Current Count Register -> 16 bit**

→ IF Program == 1  
Register is in Programming condition or IDLE state. internal signal.

→ IF Program == 0  
Register is in ACTIVE State.

→ IF Program == 1  
IF FF == 1  
Registers load DB7-DB0 to upper byte  
IF FF == 0  
Registers load DB7-DB0 to lower byte

## 16 - Bit Temporary Registers



**NOTE:**  
IdcurrAddrTemp , IdcurrWordTemp ,  
IdtempCurrAddr , IdtempWordCount -> Internal  
Signals declared in the code.

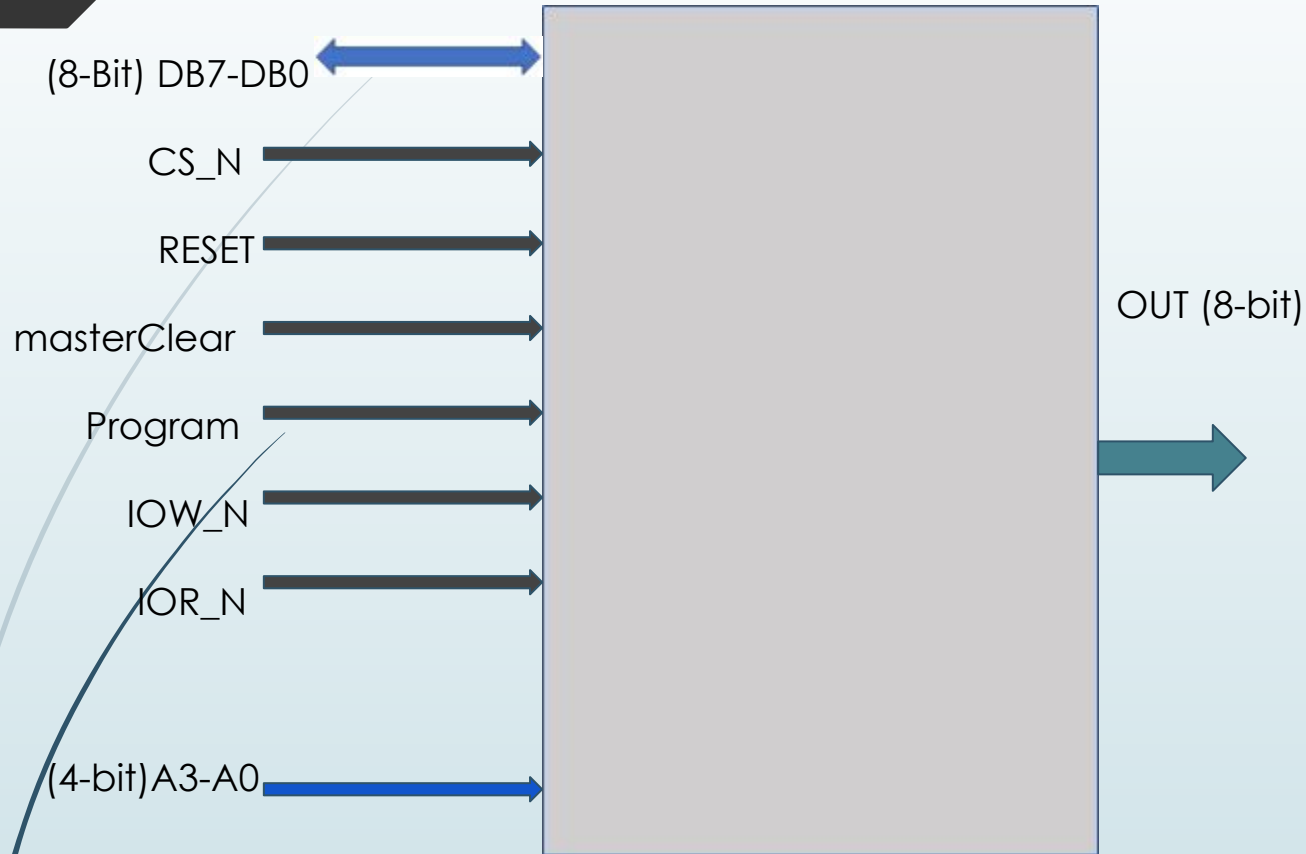
The shown Block Diagram shows the implementation of the Temporary Registers

**Temporary Address Register - 16 bit**

**Temporary Word Count Register - 16 bit**

- IF IdcurrAddrTemp == 1  
the address stored in temporary is loaded into current address register for new transfer
- IF IdcurrWordTemp == 1  
the word count in temporary is loaded into current word count register for new transfer.
- IF IdtempCurrAddr == 1  
Current address is loaded  
also LSB 8 bits will be loaded into the A7-A0 as output  
  
and decrement or increment according to the Mode[5] bit.
- IF IdtempWordCount == 1  
Current Word is loaded and decremented

## 8-Bit Registers



The shown block diagram is the implementation of the

**Command Register -> 8 bit**

**Mode Register -> 8 bit**

**Request Register -> 8 bit**

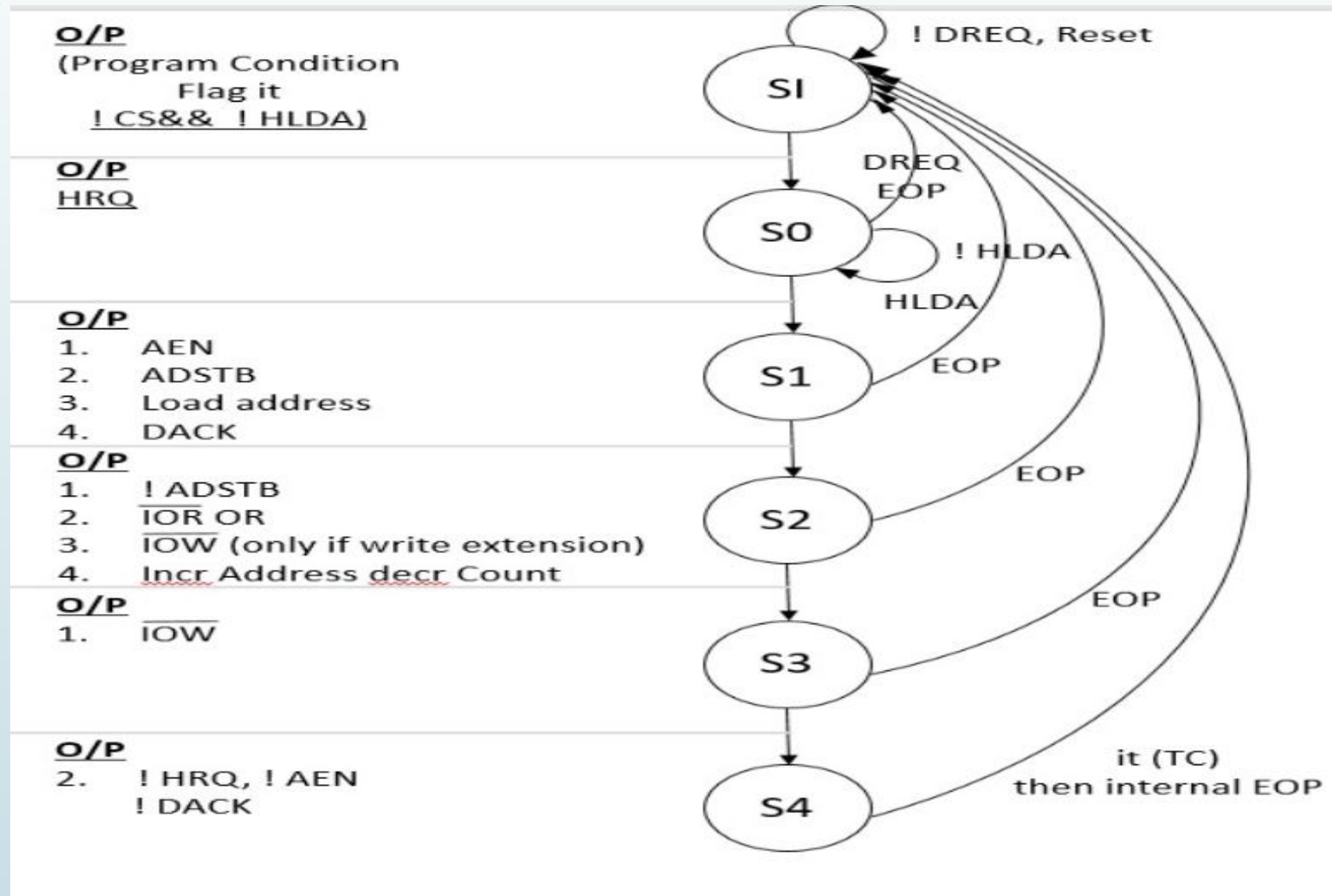
**Temporary Register -> 8 bit**

**Mask Register -> 8 bit**

→ IF Program == 1  
Register is in Programming condition or IDLE state. internal signal.

→ IF Program == 0  
Register is in ACTIVE State.

# DMA timing control and FSM



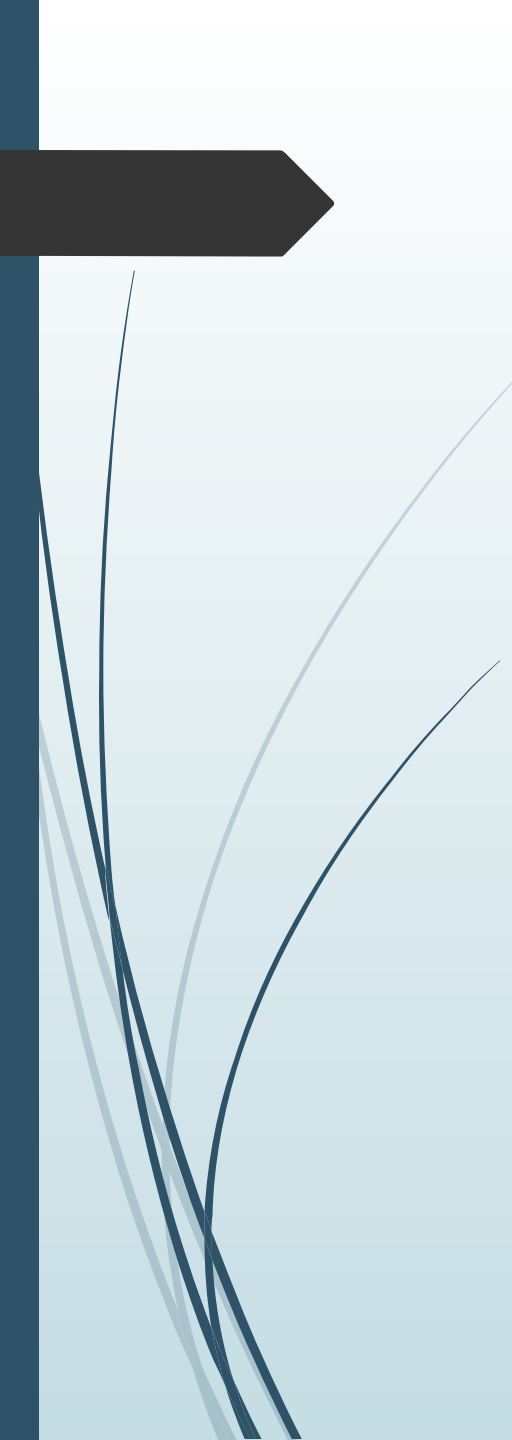
# DMA Working in IDLE Cycle

- ❑ DMA works in two Modes the Idle mode and the Active mode.
- ❑ It will be in the Idle state SI until it gets a request from any of the I/O devices.
- ❑ In the Idle mode, DMA enters into a program condition if the CS\_N and HLDA signals are low.
- ❑ In the program condition, the CPU will Chip Select DMA as an IO device, program the DMA registers for data transfers.
- ❑ Once the DMA gets a valid Request (DREQ), it sends a hold request (HRQ) signal to the CPU for bus ownership and moves to S0 state.
- ❑ DMA can still be programmed at this stage by the CPU until it gets a HLDA from CPU which indicates that the CPU has now relinquished bus ownership. Once HLDA is received by DMA it enters into active cycle.

# DMA Working in ACTIVE Cycle

- ❑ In S1 state, DMA will also load the address stored in the Current Address register on the address line .
- ❑ As the address lines are 8 bits and we have 16 bits address DMA will use the 8-bit wide data lines to send the MSB address bits A15-A8. For this DMA asserts ADSTB (Address Strobe) signal.
- ❑ In state S2, DMA will have the 16 bits address to be sent. Internal signals are used to update the count and the address register is made for generating address for next transfer .
- ❑ DMA will assert control signals IOW and MEMR for read operation, and IOR and MEMW for an Extended Write operation. Extended write is a feature where DMA does an early Write while in S2 state.
- ❑ In State S3, there will be write operations and control signals IOR and MEMW will be asserted.



- 
- ❑ In S4 state, DACK and HRQ signal will be de-asserted as it is a single transfer mode and 8 bits of data were successfully transferred.
  - ❑ Apart from this, an EOP output will be generated based on the status register bits are set based on Terminal Count (a condition where all transfers are complete).
  - ❑ After S4 state, the DMA moves back to S1 state.
  - ❑ In between any state DMA can move to S1 state if EOP input is asserted thus terminating active transfer.
  - ❑ In the Active mode, the DMA will first send the Address Enable Signal/ AEN at the same time also send DACK signal to acknowledge the I/O device and begin transfer which is the S1 state of FSM.

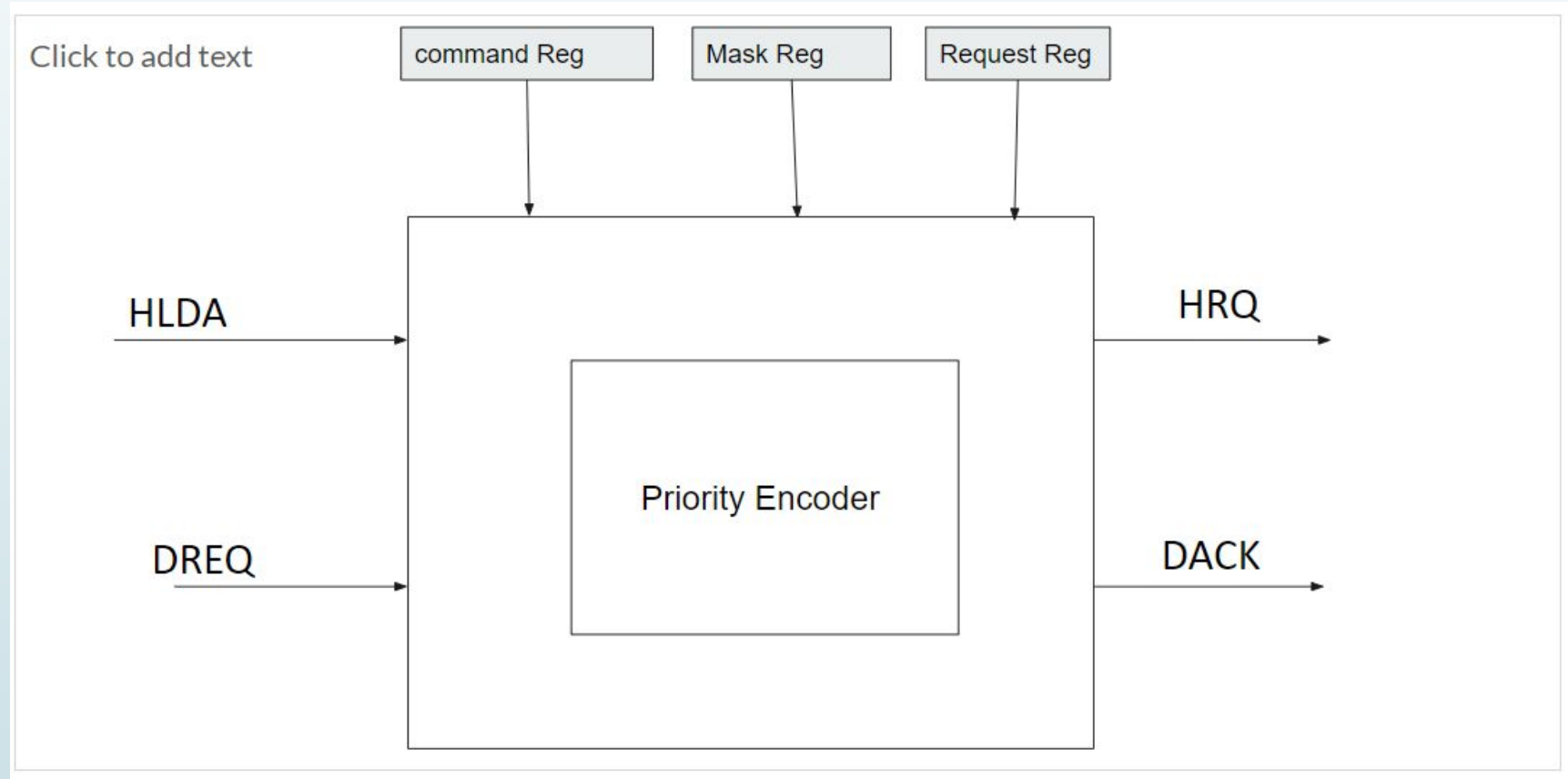
[illegible]



# Priority Logic

- ❑ 8237A has two types of priority encoding available as software commands , fixed and rotating priority.
- ❑ Fixed priority – fixes the channel in priority order based on descending value of their channel number. Channel 0 has highest priority and Channel 3 has lowest priority.
- ❑ Rotating priority – the last channel which is served has lowest priority channel with the others rotating accordingly.
- ❑ Command Register , Mask Register , Request Register are used by CPU to communicate the Channel information to Priority logic.

# Priority Logic





# Challenges Faced and Learnings

- Thorough understanding of the datasheet for correct design implementation.
- Dividing the design implementation into submodules between a team of four and still maintaining code coherency using Git.
- Passing Clock and Reset signals from top level interface to DMA top, submodules and internal interfaces.
- Using interfaces with mod ports for all connections.
- Modeling Bi-directional signals.
- Maximum usage of System Verilog constructs like interfaces, modports, priority, unique, reverse case, always\_comb and always\_ff.
- Implementing independent polarity control for (DREQ) Request and (DACK) Acknowledge signals.
- Debugging RTL design issues using a testbench.



# Future scope of improvement

- ❑ Currently, we have only implemented Single transfer mode. We can improve the design to support Block transfer, Demand transfer and Cascade transfer (multiple instances of DMA top required).
- ❑ The design can be extended to support Verify Pseudo Read/Write transfers.
- ❑ The design can be extended to support memory to memory transfers using block transfer mode.
- ❑ The design can be extended to support more channels by parameterizing the DREQ and DACK signals and Channel specific registers, thus also making the design run time configurable.
- ❑ The design can be improved to strictly follow the timing values (by using parameters) given in the datasheet.
- ❑ Assertions can be binded to the design at critical points to ensure that the unknown ('x/z') inputs provided to the design are caught at that point.



# References and Project URL

- Intel 8237A High Programmable DMA controller Datasheet provided by Prof. Faust.
- DMA controller lecture slides from ECE 585 class taught by Prof. Faust.  
<http://web.cecs.pdx.edu/~faustm/ece485/lectures/2-BasicIO.pdf>
- Project URL:  
<https://github.com/srishis/DMA8237A>



**Thank you**