

# Outline

1. DSP project info
2. Analog-to-digital converters
3. Digital-to-analog converters
4. 2-dimensional register memory

# DSP Project

**Reminder:** Project proposal is due Friday, October 10, 2008 by 5pm.

→ Lab book extensions possible.

**Budget change:** \$200 for project.

**Spec. modification:** Extra trigger input for starting the function generator

→ trigger sets  $t=0$ .

**Basic guideline for project proposal** (also see *Project Guidelines* document):

You must convince your reader that your design concept will work well enough that it deserves to be funded. This means that your design has reached the point that you can draw up a budget, and the basic design algorithm has been worked out. *Your proposal must include a budget which is as specific as possible and an expected timeline.*

**Free parts:** Surplus parts from last year's project will be available on a first-come-first-serve basis during the Wednesday, October 15, 2008 lab period.

# Analog ↔ Digital

As physicists, we know that:

- We live in an **analog** world of continuously varying signals.
- Almost all physical quantities (observables) are continuous in nature.

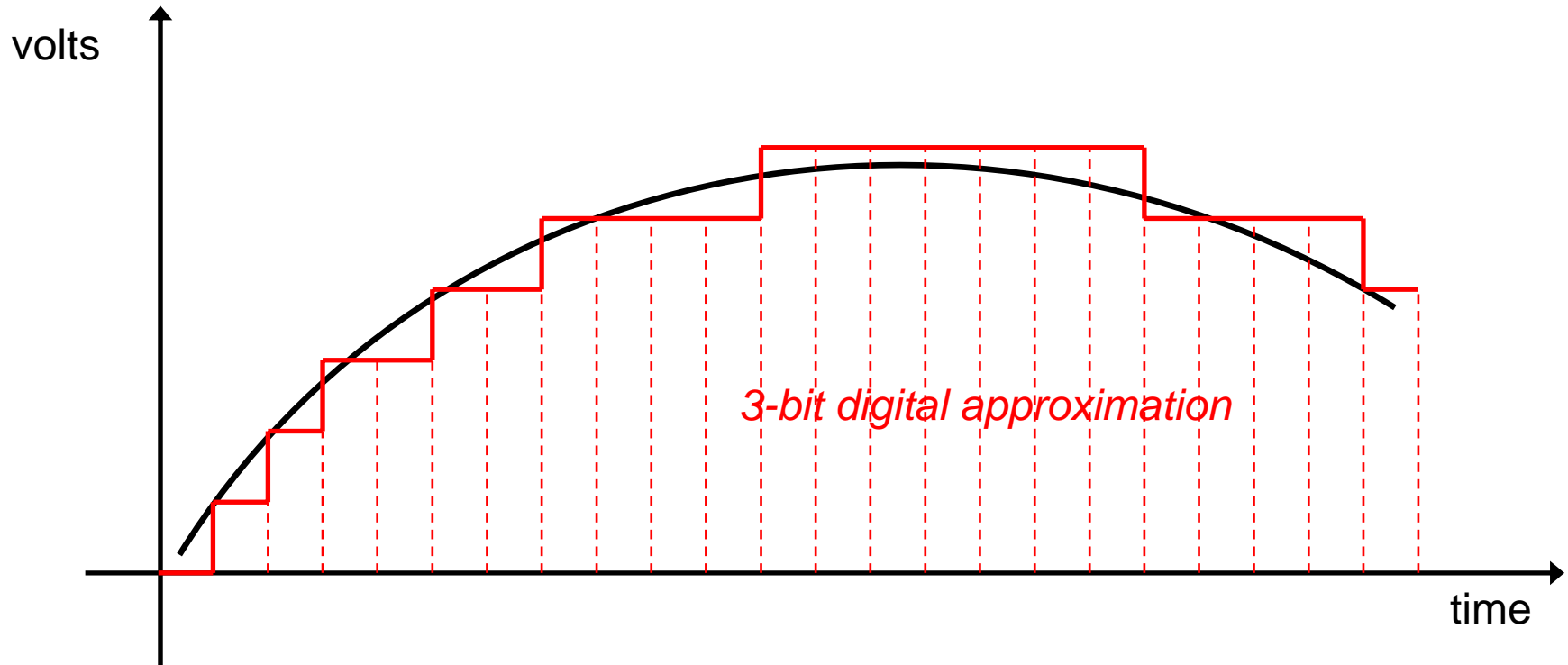
As electronics designers, we know that:

- Digital electronics is very **powerful**, **cheap**, and relatively **easy** to design (at least compared to analog circuits).
- Digital electronics only works with **digital signals**.

THEREFORE ...

If we're ever going to make anything useful, we need to find a way to convert (or approximate) an ANALOG signal to (by) a sequence of digital-binary numbers, and vice versa.

# Analog → Digital



**Algorithm:** → At each clock cycle, round your analog voltage to the nearest digital value.

→ The size of a digital step is defined by  $V_{\text{ref}}/2^n$  for an n-bit converter.

# Shannon-Nyquist Sampling Theorem

## ***THEOREM:***

A continuous-time finite bandwidth signal can be exactly reconstructed from its samples if the sampling frequency is greater than 2 times the signal bandwidth  $B$ , where  $B$  is largest (non-zero) frequency component of the signal.

$F=2B$  is referred to as the Nyquist frequency (the lowest possible sampling frequency).

## ***Practical Considerations:***

- Any finite duration signal has  $B \rightarrow +\infty$ , so exact mathematical application of the theorem is impossible.
- The theorem indicates the frequency scale that one should use in order to usefully sample a signal → ***always use a sampling rate which is greater than twice the highest frequency component of “reasonable amplitude”.***

# Flash ADCs

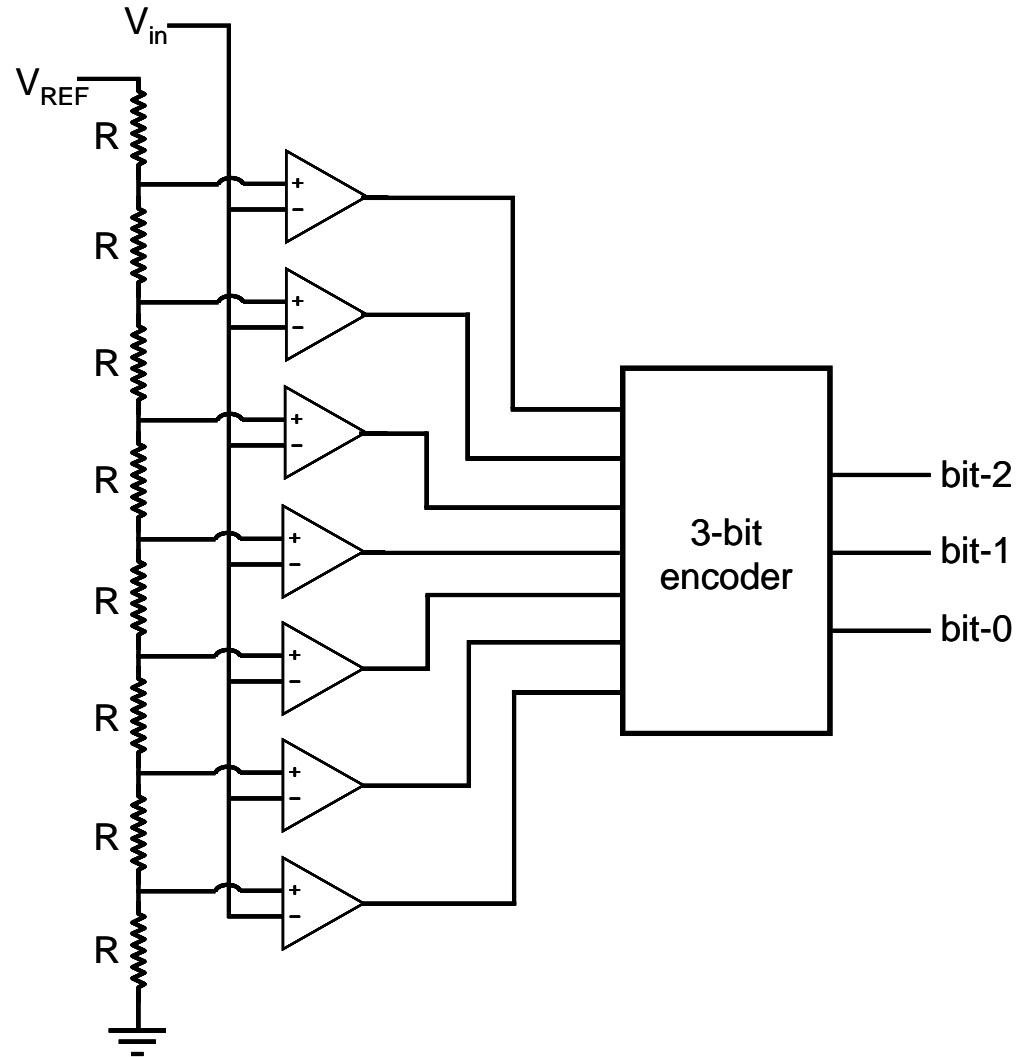
The fastest (and most expensive)  $n$ -bit ADCs use  $2^n - 1$  comparators to determine which of the  $2^n$  numbers the analog input is closest to.

8-bit ADC: 255 comparators.

12-bit ADC: 4096 comparators

In general, higher bit resolution results in a slower ADC (and more expensive).

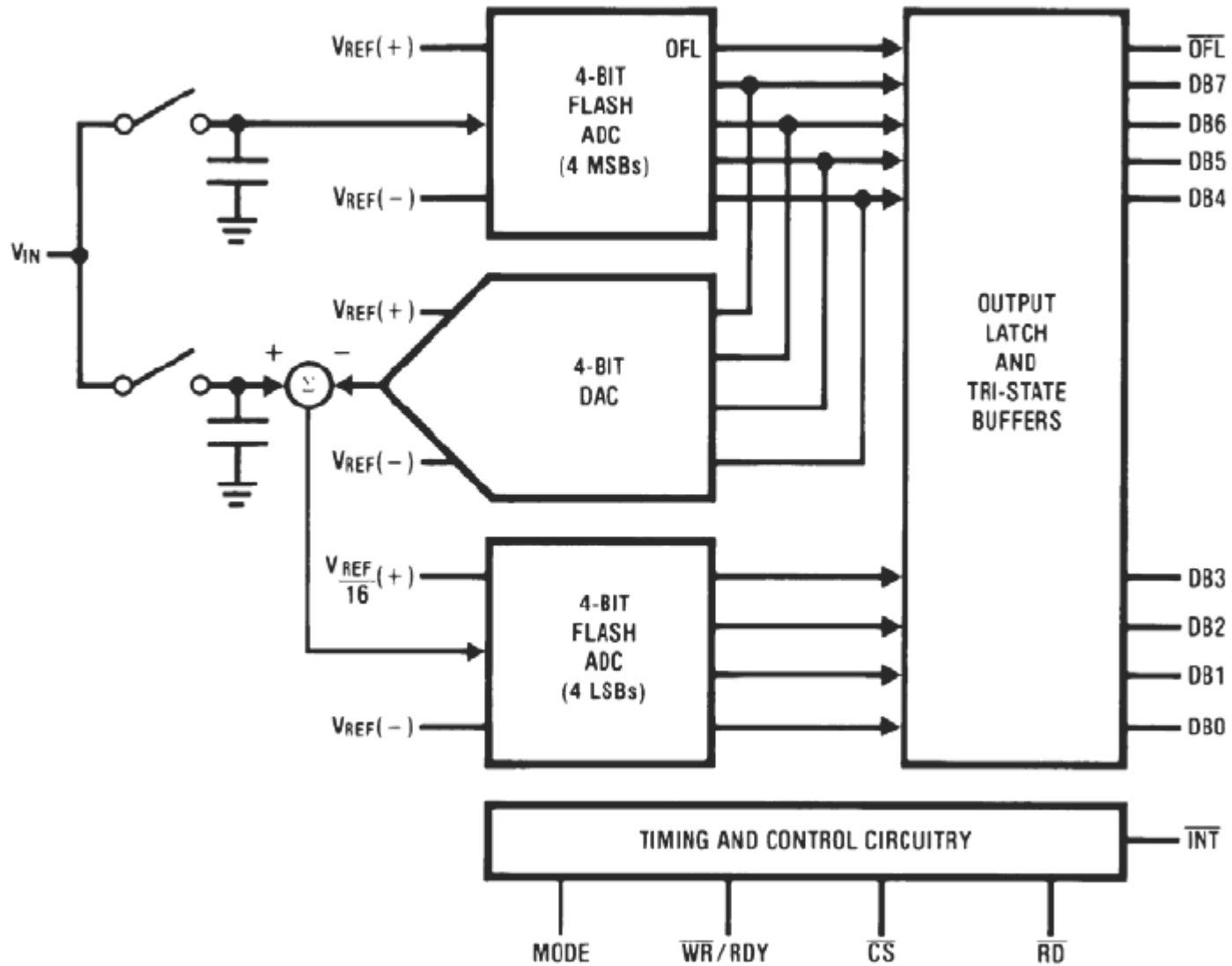
Most digital oscilloscopes use an 8-bit ADC.



# ADC0820 & TLC5510A: Half-Flash ADC (I)

- In order to keep the number of comparators small, it holds the input voltage, and converts it in steps:
  - Converts the upper four bits with a 4-bit ADC.
  - Converts the digitized value back into an analog value with a Digital-to-Analog Converter (DAC).
  - Subtracts this from the input to generate the smaller, difference voltage.
  - Finally, it uses a 2<sup>nd</sup> 4-bit ADC to convert the lower 4-bits.
  
- The entire process takes less than 800 ns when operating off the internal timing of the ADC0820 (RD mode) and about 150 ns for a TLC5510A (20 MHz clock).

# ADC0820: Half-Flash ADC (II)



[figure from the National Semiconductor ADC0820 datasheet]



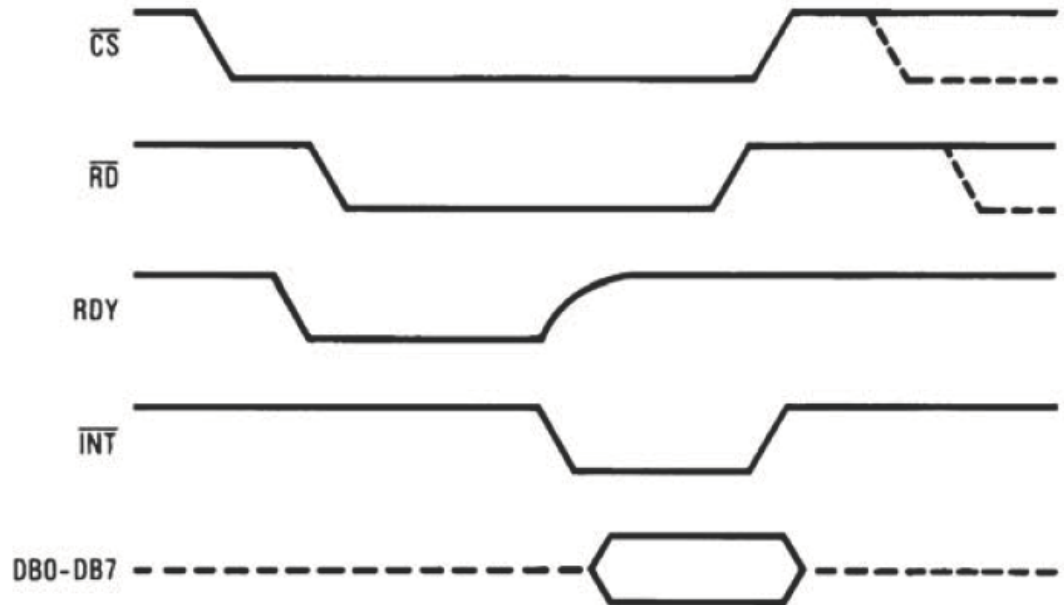
# ADC0820: Half-Flash ADC (III)

## Functional Table

Pin	Name	Function
1	VIN	Analog input; range $GND \leq V_{IN} \leq V_{CC}$
2-5	DB0-DB3	TRI-STATE data outputs; bit 0 (LSB) to bit 3
6	WR / RDY	<b>WR-RD Mode - WR:</b> With CS low, the conversion is started on the falling edge of WR. <b>RD Mode - RDY:</b> RDY will go low after the falling edge of CS; RDY will go TRI-STATE when the result of the conversion is strobed into the output latch.
7	MODE	Select mode:    LOW = <b>RD Mode</b> HIGH = <b>WR-RD Mode</b>
8	RD	<b>WR-RD Mode</b> With CS low, the TRI-STATE data outputs (DB0-DB7) will be activated when RD goes low. <b>RD Mode</b> With CS low, the conversion will start with RD going LOW; also RD will enable the TRI-STATE data outputs at the completion of the conversion. RDY going TRI-STATE and INT going low indicates the completion of the conversion.
9	INT	INT going LOW indicates that the conversion is completed and the data result is in the output latch. INT is reset by rising edge on RD or CS.
11	$V_{REF(-)}$	Bottom of resistor ladder; range: $GND \leq V_{REF(-)} \leq V_{REF(+)}$
12	$V_{REF(+)}$	Top of resistor ladder; range: $V_{REF(-)} \leq V_{REF(+)} \leq V_{CC}$
13	CS	CS must be low for the RD or WR to be recognized.
14-17	DB4-7	TRI-STATE data output—bits 4-7
18	OFL	Overflow—If the analog input is higher than the $V_{REF(+)}$ , OFL will be LOW at the end of conversion. Can be used to cascade.

# ADC0820: Half-Flash ADC (IV)

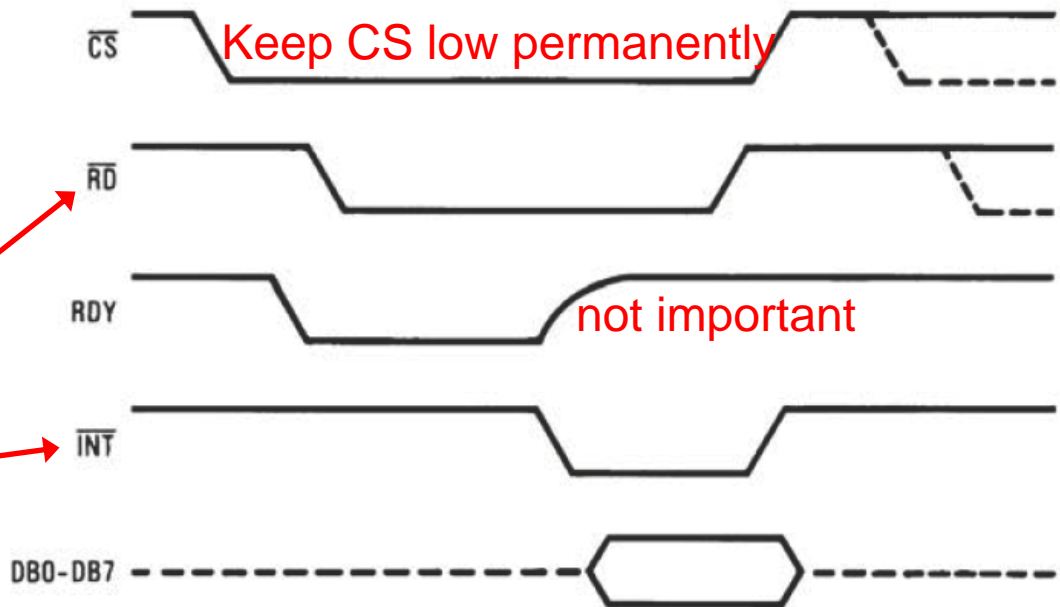
- In RD mode, the RD# line going LOW initiates the conversion.
- When the conversion is complete, the INT# line goes LOW & the data is latched into output buffers.
- The output buffers will be put in a Z state when WR# goes LOW until the INT# line goes LOW.



# ADC0820: Half-Flash ADC (IV)

- In RD mode, the RD# line going LOW initiates the conversion.
- When the conversion is complete, the INT# line goes LOW & the data is latched into output buffers.
- The output buffers will be put in a Z state when WR# goes LOW until the INT# line goes LOW.

These lines are  
IMPORTANT



# TLC5510A: Basic Operation

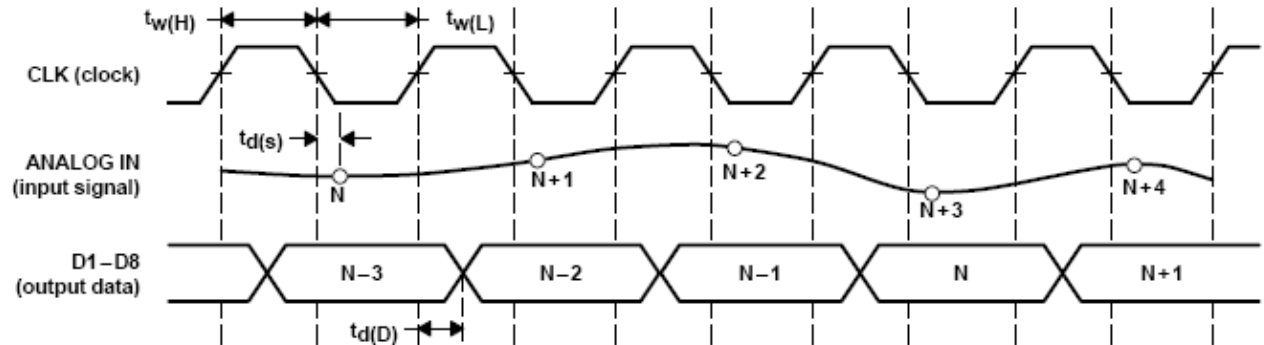
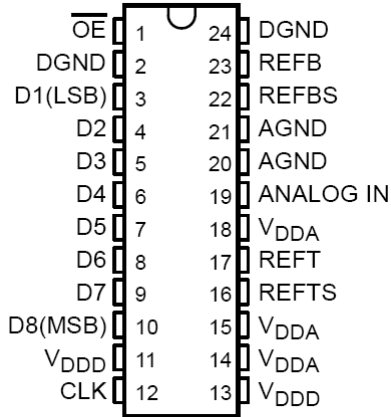


Figure 1. I/O Timing Diagram

## Terminal Functions

TERMINAL NAME	NO.	I/O	DESCRIPTION
AGND	20, 21		Analog ground
ANALOG IN	19	I	Analog input
CLK	12	I	Clock input
DGND	2, 24		Digital ground
D1–D8	3–10	O	Digital data out. D1 = LSB, D8 = MSB
$\overline{OE}$	1	I	Output enable. When $\overline{OE}$ = low, data is enabled. When $\overline{OE}$ = high, D1–D8 is in high-impedance state.
$V_{DDA}$	14, 15, 18		Analog supply voltage
$V_{DDD}$	11, 13		Digital supply voltage
REFB	23	I	Reference voltage in bottom
REFBS	22		Reference voltage in bottom. When using the TLC5510 internal voltage divider to generate a nominal 2-V reference, REFBS is shorted to REFB (see Figure 3). When using the TLC5510A, REFBS is connected to ground.
REFT	17	I	Reference voltage in top
REFTS	16		Reference voltage in top. When using the TLC5510 internal voltage divider to generate a nominal 2-V reference, REFTS is shorted to REFT (see Figure 3). When using the TLC5510A, REFTS is connected to $V_{DDA}$ .

# Mixed Signal Design

## APPLICATION INFORMATION

The following notes are design recommendations that should be used with the device.

- External analog and digital circuitry should be physically separated and shielded as much as possible to reduce system noise.
- RF breadboarding or printed-circuit-board (PCB) techniques should be used throughout the evaluation and production process. Breadboards should be copper clad for bench evaluation.
- Since AGND and DGND are connected internally, the ground lead in must be kept as noise free as possible. A good method to use is twisted-pair cables for the supply lines to minimize noise pickup. An analog and digital ground plane should be used on PCB layouts when additional logic devices are used. The AGND and DGND terminals of the device should be tied to the analog ground plane.
- $V_{DDA}$  to AGND and  $V_{DDD}$  to DGND should be decoupled with 1- $\mu$ F and 0.01- $\mu$ F capacitors, respectively, and placed as close as possible to the affected device terminals. A ceramic-chip capacitor is recommended for the 0.01- $\mu$ F capacitor. Care should be exercised to ensure a solid noise-free ground connection for the analog and digital ground terminals.
- $V_{DDA}$ , AGND, and ANALOG IN should be shielded from the higher frequency terminals, CLK and D0–D7. When possible, AGND traces should be placed on both sides of the ANALOG IN traces on the PCB for shielding.
- In testing or application of the device, the resistance of the driving source connected to the analog input should be 10  $\Omega$  or less within the analog frequency range of interest.

# Digital → Analog

A Digital-to-Analog Converter (DAC) is used to convert a digital signal into an analog voltage.

## ***A DAC is useful for:***

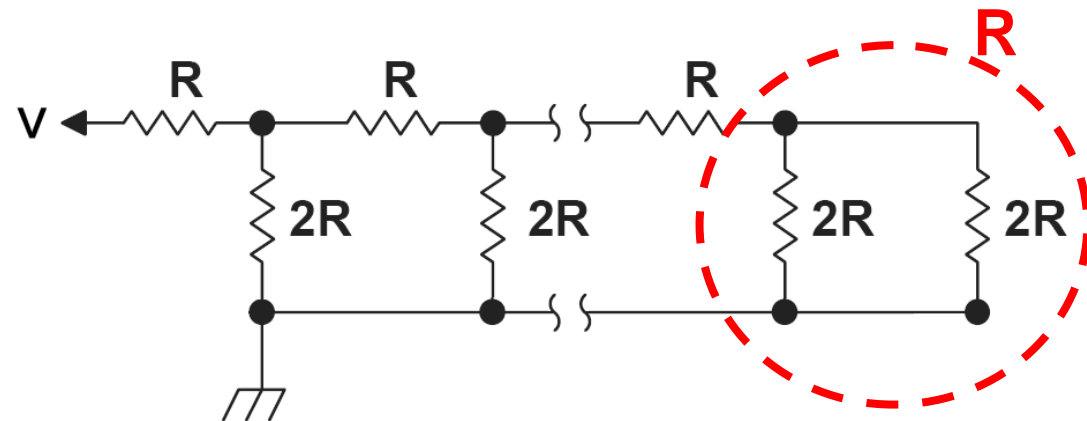
- Generating a voltage from a computer, microprocessor, or FPGA that will then control part of an experiment.
- Producing a digitally synthesized waveform (triangle, sine, or more complex).
- Converting digital music to sound, etc ...

(the CD standard is 16-bits at 44.1 KHz)

DACs are generally much faster than ADCs for a same bit resolution.

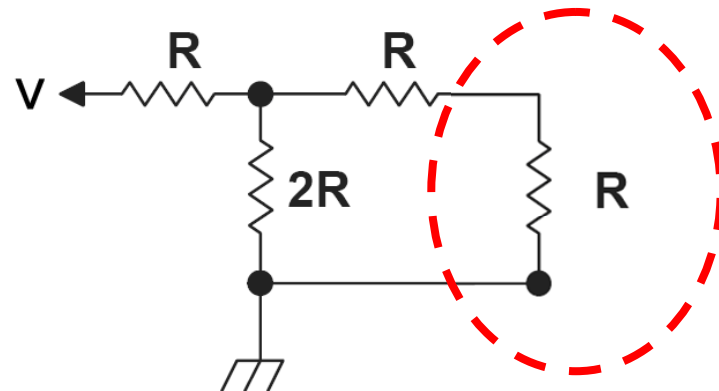
# R-2R resistor ladder (I)

- It's easy to make the DAC output voltages.
- Look from the right-hand side
  - 2 parallel resistors
  - Each with a value of  $2R$



# R-2R resistor ladder (I)

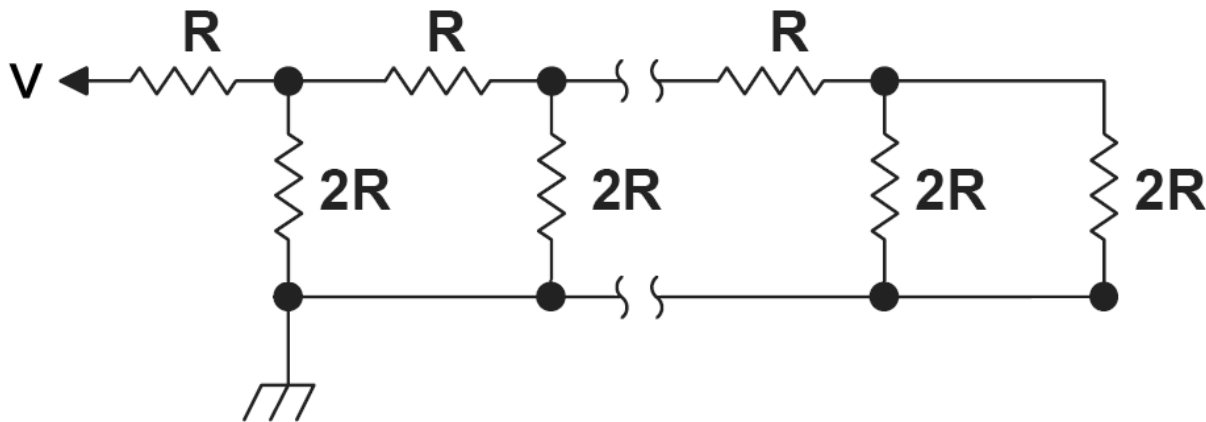
- It's easy to make the DAC output voltages.
- Look from the right-hand side
  - 2 parallel resistors
  - Each with a value of  $2R$





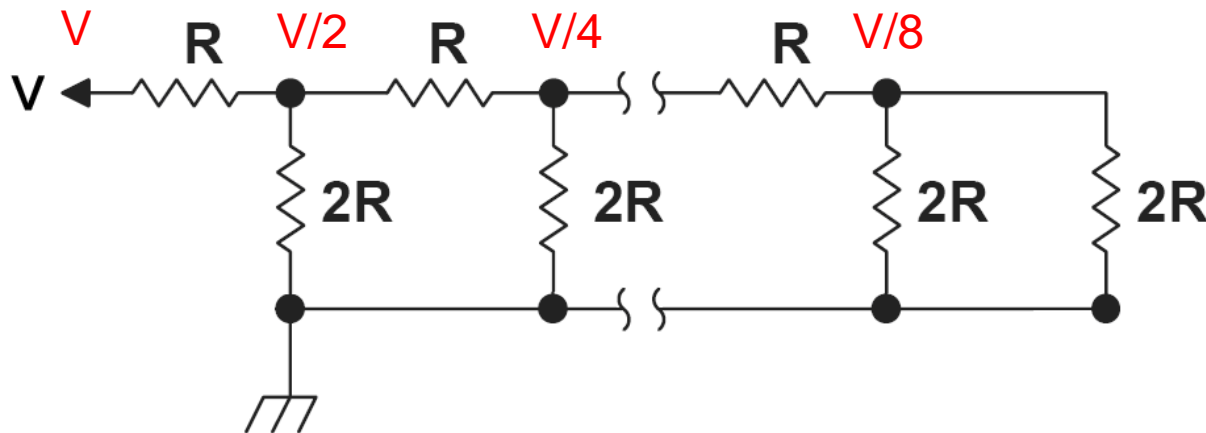
## R-2R resistor ladder (II)

- Continuing farther to the left, we find that the effective resistance to ground is  $R$  at every dot on the top line



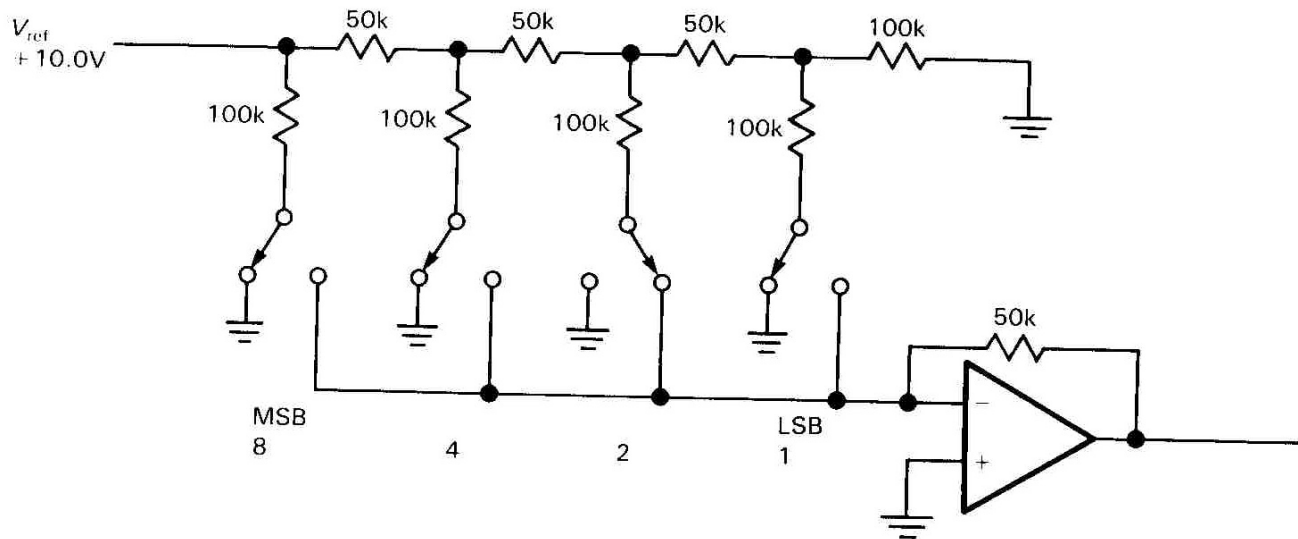
# R-2R resistor ladder (III)

- The ladder acts like a series of voltage dividers that reduces the voltage by an additional factor of 2 at each  $R$ - $2R$  junction.
- $V$  decreases by half at each connection point along the top rail.
- Thus each output voltage is related to the input voltage by a power of two.



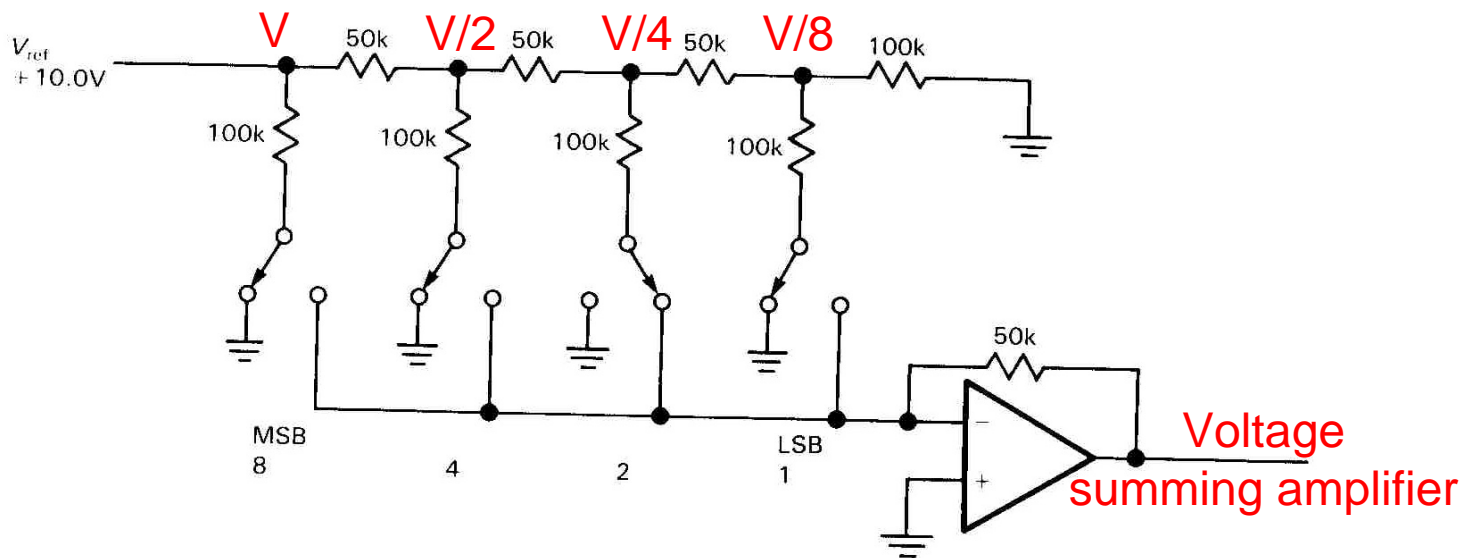
# Simple DAC

- We can generate an analog voltage by adding together the voltages represented by the various stages in the ladder.
- If we sum the ladder outputs based on a simple a binary representation in switches then we have a DAC.



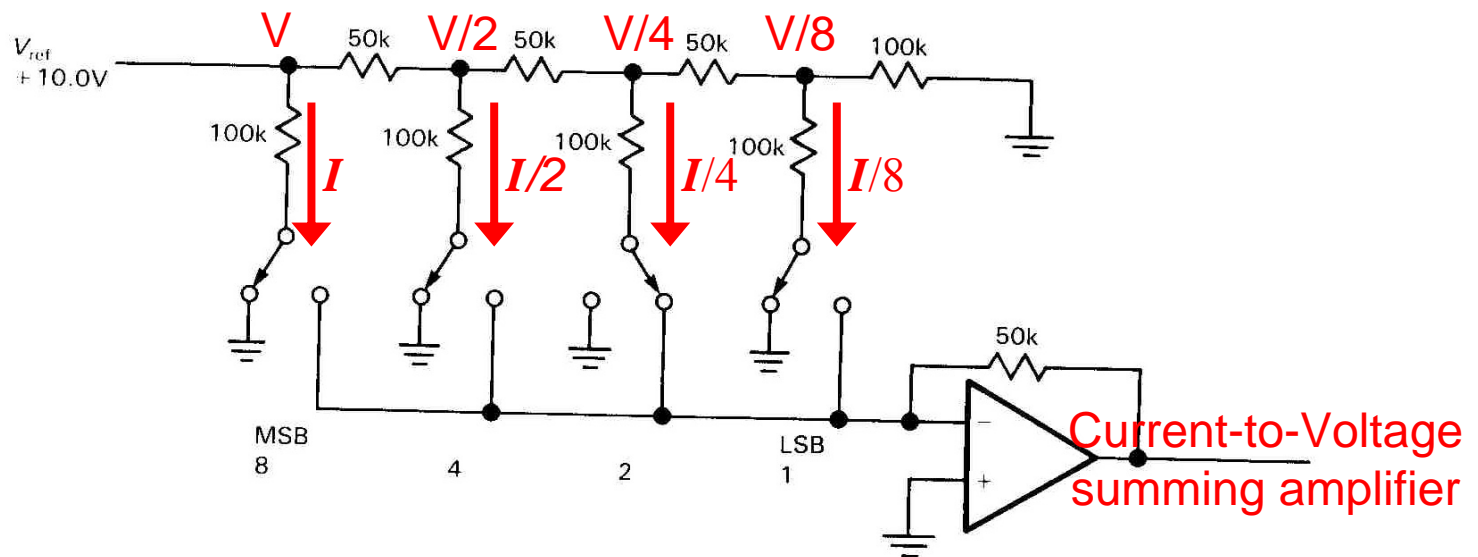
# Simple DAC

- We can generate an analog voltage by adding together the voltages represented by the various stages in the ladder.
- If we sum the ladder outputs based on a simple a binary representation in switches then we have a DAC.

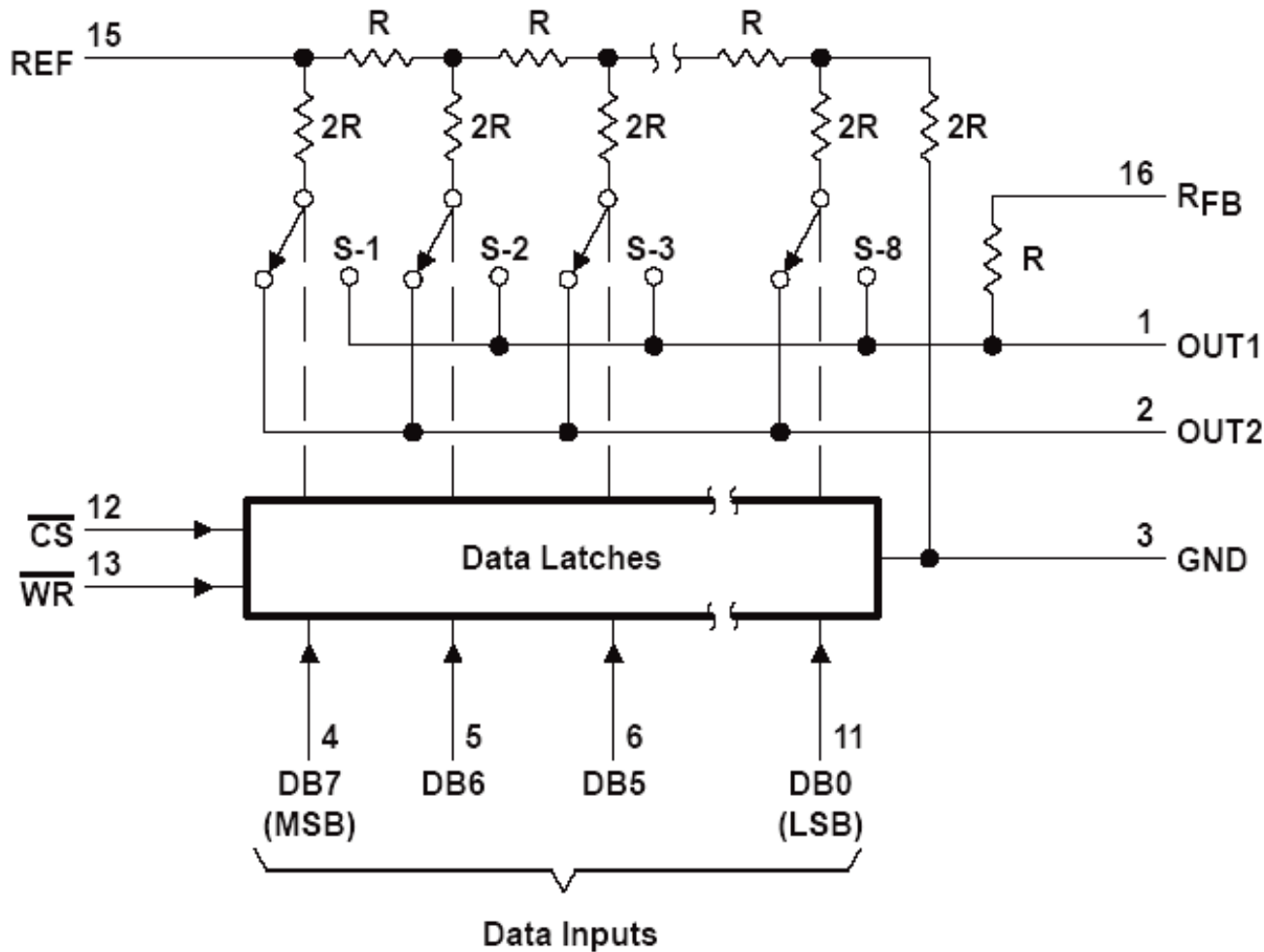


# Simple DAC

- We can generate an analog voltage by adding together the voltages represented by the various stages in the ladder.
- If we sum the ladder outputs based on a simple a binary representation in switches then we have a DAC.



# A real DAC: the TLC7524

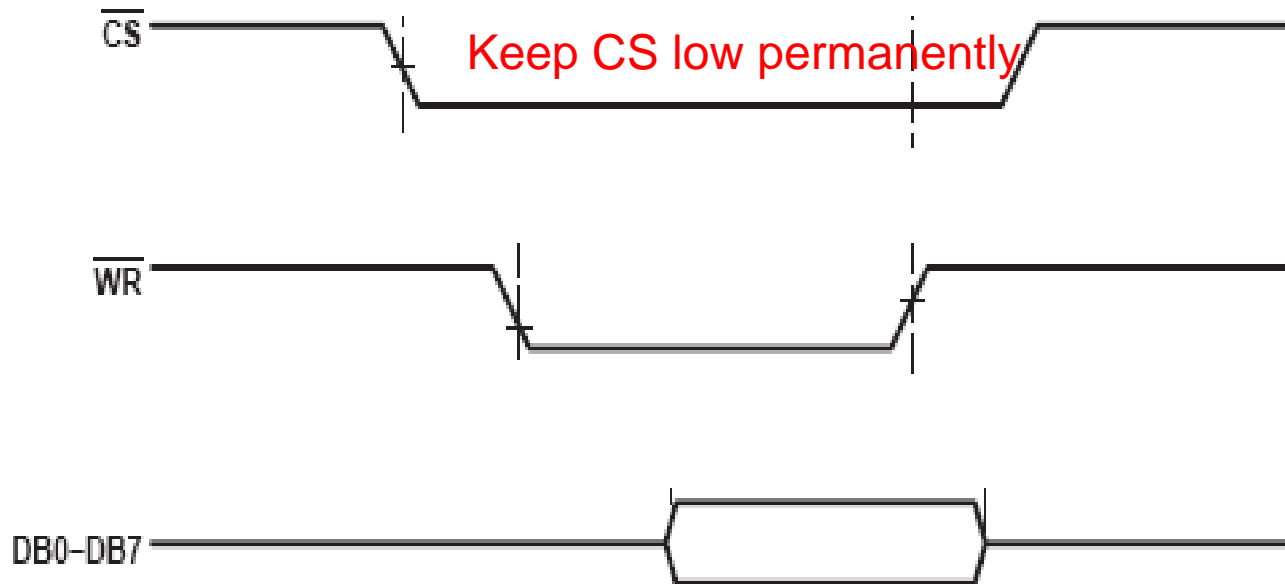


**DAC (TLC7524)**

[figure from Texas Instruments TLC7524 datasheet]

# TLC7524 timing

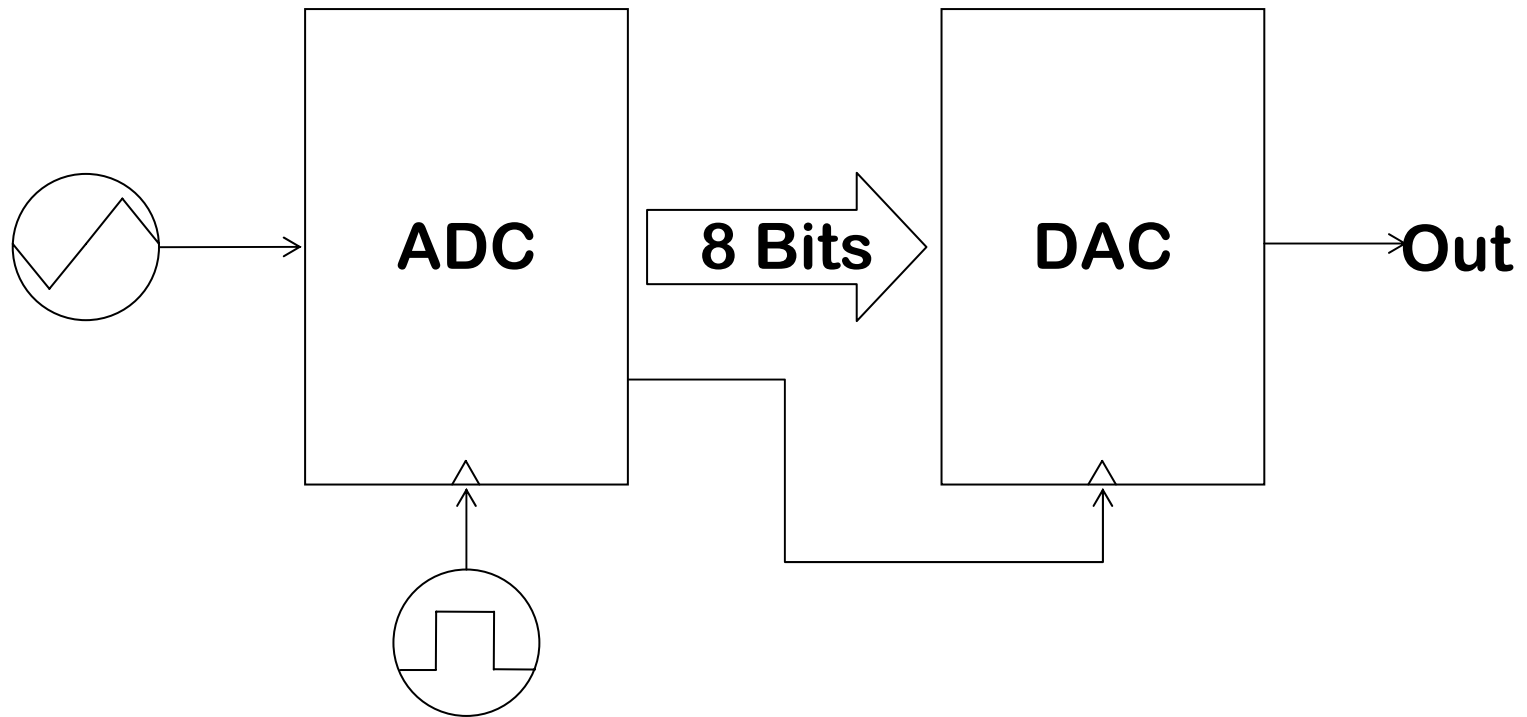
- The data on the digital inputs is sent to the analog output when WR goes LOW.
- When WR is HIGH, the digital inputs and analog output remain latched to their present values.



[figure from Texas Instruments TLC7524 datasheet]

# ADC → DAC

## Analog-to-Analog





# 2-dimensional registers with Quartus II

```
1 // This module produces a sequence of 8-bit binary numbers
2 // with a period of 1 second (based on a 50.0 MHz clock).
3 module TwoD_Register(clock,output_signal);
4     input clock; // input clock at 50 MHz
5     output reg [7:0] output_signal; // output 8-bit binary number sequence
6
7     reg [25:0] frequency_counter; // frequency counter used to convert 50 MHz clock to 1 Hz clock
8     reg [5:0] i; // 5-bit (integers: 0-63) counter/register for addressing memory
9
10 // Simultaneously Declare and Load the data file "data" into 2-d register memory on FPGA
11 // This "(* ... *)" code is compiler specific to Quartus II (it is not universal Verilog).
12 // The 2-d register memory consists of 30 words which are 8-bits long each.
13     (* ram_init_file = "File_with_Data.mif" *) reg [7:0] Register_2D [0:29];
14
15 // initialize 1-d registers
16     initial
17     begin
18         i = 5'b00000;
19         frequency_counter = 26'b00000000000000000000000000000000;
20     end
21
```

# 2-dimensional registers with Quartus II

```
1 // This module produces a sequence of 8-bit binary numbers
2 // with a period of 1 second (based on a 50.0 MHz clock).
3 module TwoD_Register(clock,output_signal);
4     input clock; // input clock at 50 MHz
5     output reg [7:0] output_signal; // output 8-bit binary number sequence
6
7     reg [25:0] frequency_counter; // frequency counter used to convert 50 MHz clock to 1 Hz clock
8     reg [5:0] i; // 5-bit (integers: 0-63) counter/register for addressing memory
9
10 // Simultaneously Declare and Load the data file "data" into 2-d register memory on FPGA
11 // This "(* ... *)" code is compiler specific to Quartus II (it is not universal Verilog).
12 // The 2-d register memory consists of 30 words which are 8-bits long each.
13     (* ram_init_file = "File_with_Data.mif" *) reg [7:0] Register_2D [0:29];
14
15 // initialize 1-d registers
16     initial
17     begin
18         i = 5'b00000;
19         frequency_counter = 26'b00000000000000000000000000000000;
20     end
21
```

Declare 2-d register  
&  
Initialize with data in file.

# Using a 2-d Register in Verilog

```
21
22 // This 50 MHz clock counter counts up to 50e6 in 1 seconds and then resets
23 // and increases in the "i" counter by 1 ("i" clock is effectively 1 Hz).
24 always@(posedge clock)
25     begin
26
27         // increase 50 MHz frequency_counter by 1.
28         frequency_counter <= frequency_counter + 1;
29
30         // if "frequency_counter" is equal to 50 million,
31         // then reset frequency_counter to ZERO
32         // and send the 8-bit word corresponding to Register_2D[i] to the output
33         // and increase the "i" counter by ONE.
34         if (frequency_counter == 26'b10111110101111000010000000)
35             begin
36                 frequency_counter <= 26'b0000000000000000000000000000;
37                 output_signal <= Register_2D[i];
38                 i <= i + 5'b00001;
39                 if (i == 5'b10000) i <= 5'b00000; // reset "i" counter to ZERO if it becomes 32.
40             end
41         end
42
43     endmodule
```

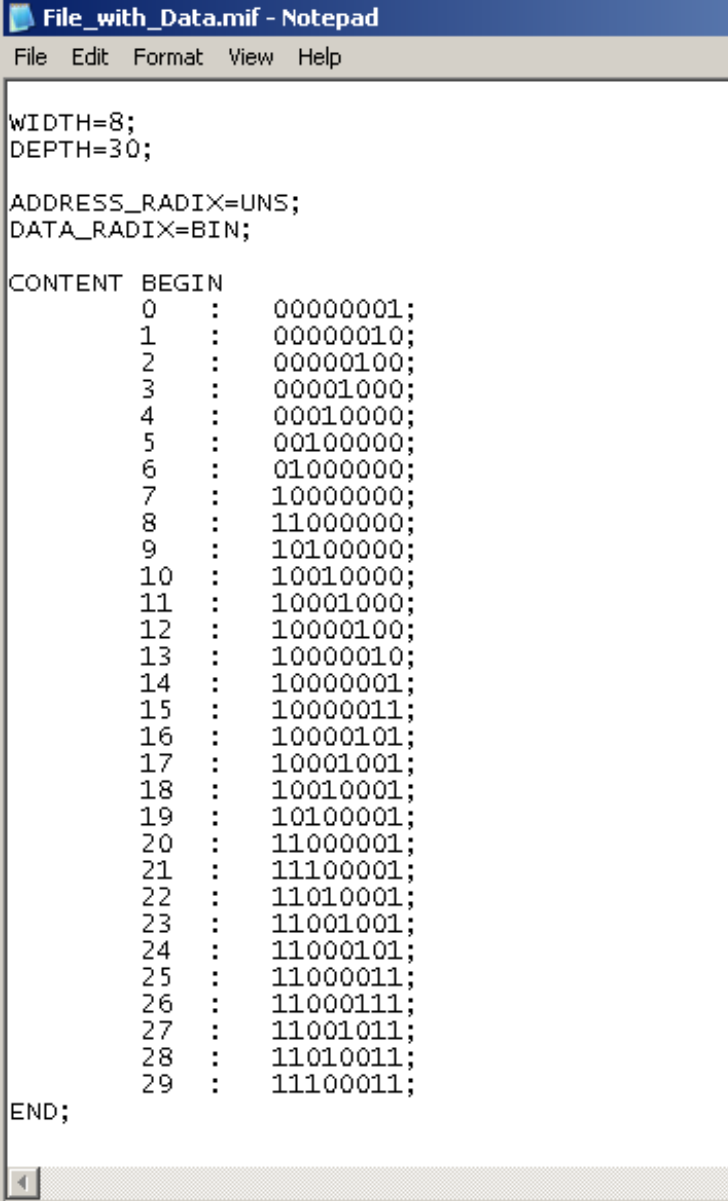
# Using a 2-d Register in Verilog

```
21
22 // This 50 MHz clock counter counts up to 50e6 in 1 seconds and then resets
23 // and increases in the "i" counter by 1 ("i" clock is effectively 1 Hz).
24 always@(posedge clock)
25     begin
26
27         // increase 50 MHz frequency_counter by 1.
28         frequency_counter <= frequency_counter + 1;
29
30         // if "frequency_counter" is equal to 50 million,
31         // then reset frequency_counter to ZERO
32         // and send the 8-bit word corresponding to Register_2D[i] to the output
33         // and increase the "i" counter by ONE.
34         if (frequency_counter == 26'b10111110101111000010000000)
35             begin
36                 frequency_counter <= 26'b0000000000000000000000000000;
37                 output_signal <= Register_2D[i];
38                 i <= i + 5'b00001;
39                 if (i == 5'b10000) i <= 5'b00000; // reset "i" counter to ZERO if it becomes 32.
40             end
41         end
42
43     endmodule
```

Reads the 8-bit word at address "i" of "Register\_2D"  
&  
sends it to the 8-bit 1-d register "output\_signal".

# Generating the *Memory Initialization File* (I)

You can write the file yourself or ...



```
File_with_Data.mif - Notepad
File Edit Format View Help

WIDTH=8;
DEPTH=30;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
    0 : 00000001;
    1 : 00000010;
    2 : 00000100;
    3 : 00001000;
    4 : 00010000;
    5 : 00100000;
    6 : 01000000;
    7 : 10000000;
    8 : 11000000;
    9 : 10100000;
   10 : 10010000;
   11 : 10001000;
   12 : 10000100;
   13 : 10000010;
   14 : 10000001;
   15 : 10000011;
   16 : 10000101;
   17 : 10001001;
   18 : 10010001;
   19 : 10100001;
   20 : 11000001;
   21 : 11100001;
   22 : 11010001;
   23 : 11001001;
   24 : 11000101;
   25 : 11000011;
   26 : 11000111;
   27 : 11001011;
   28 : 11010011;
   29 : 11100011;

END;
```

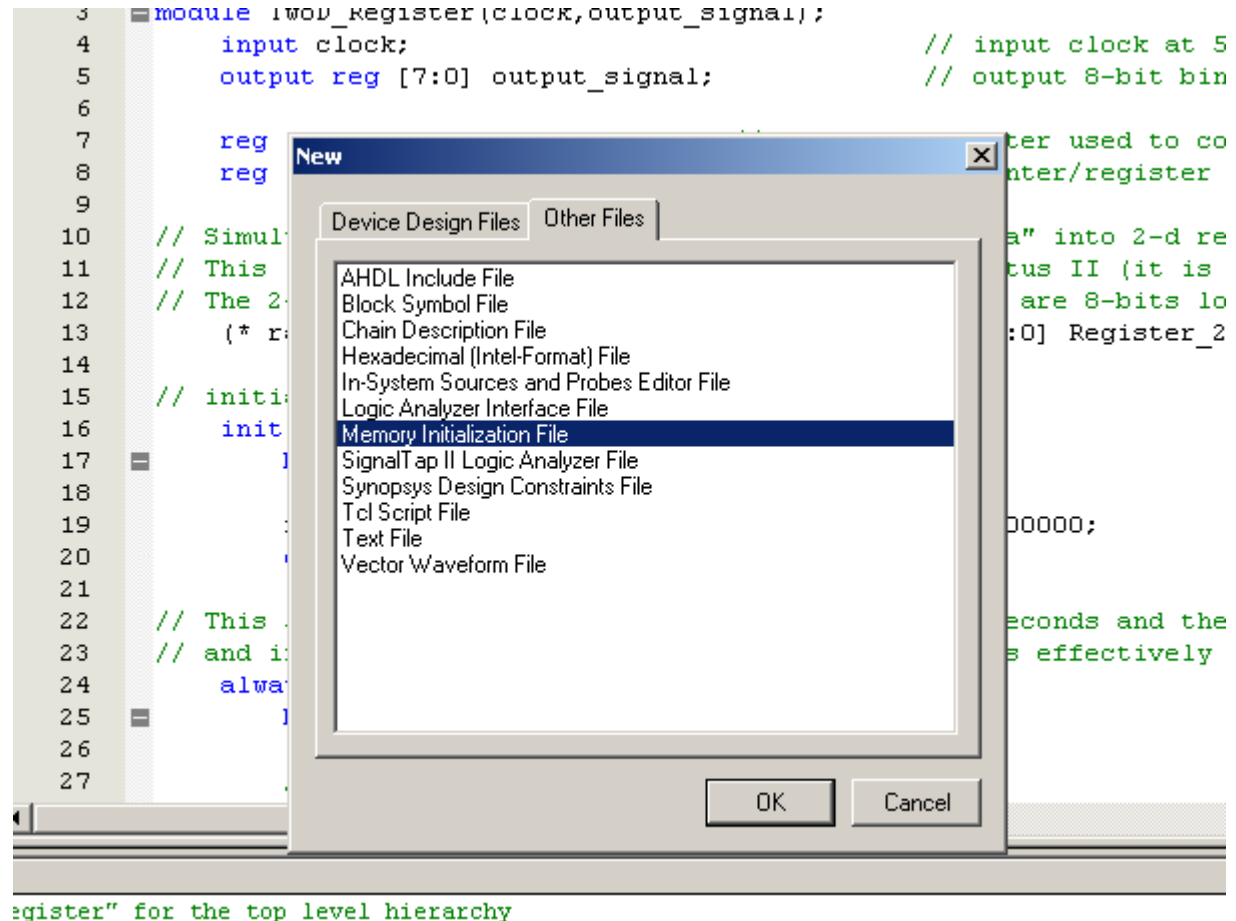
# Generating the *Memory Initialization File* (II)

You can use Quartus II to generate the file using the Memory Editor:

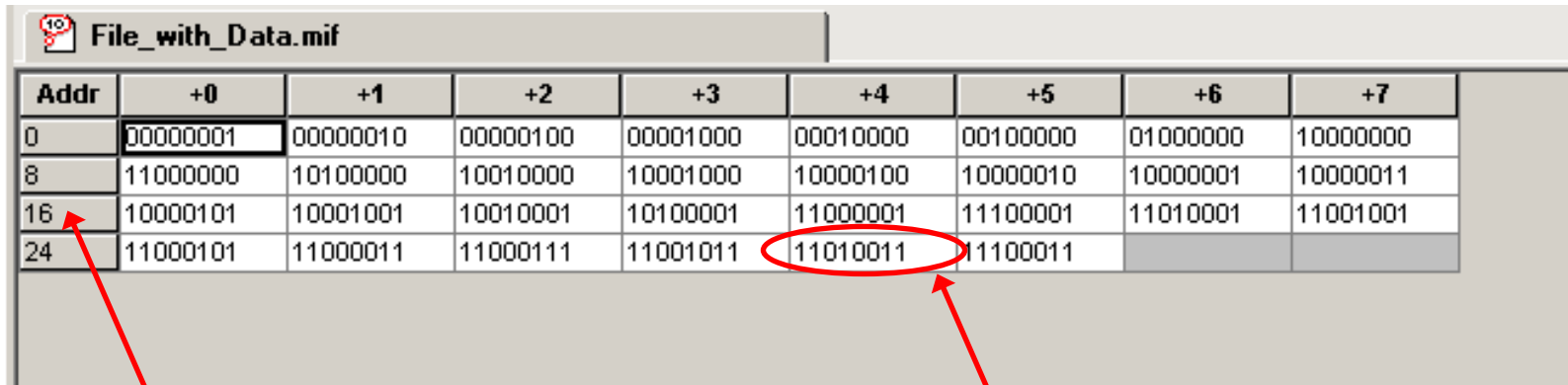
Select: New File

→ Other Files

→ Memory Initialization File



# Memory Initialization File Editor



Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	00000001	00000010	00000100	00001000	00010000	00100000	01000000	10000000
8	11000000	10100000	10010000	10001000	10000100	10000010	10000001	10000011
16	10000101	10001001	10010001	10100001	11000001	11100001	11010001	11001001
24	11000101	11000011	11000111	11001011	11010011	11100011		

word memory address

8-bit word  
(enter by hand)

**Right-click** to enter type memory information: integer, hexadecimal, binary, etc ...

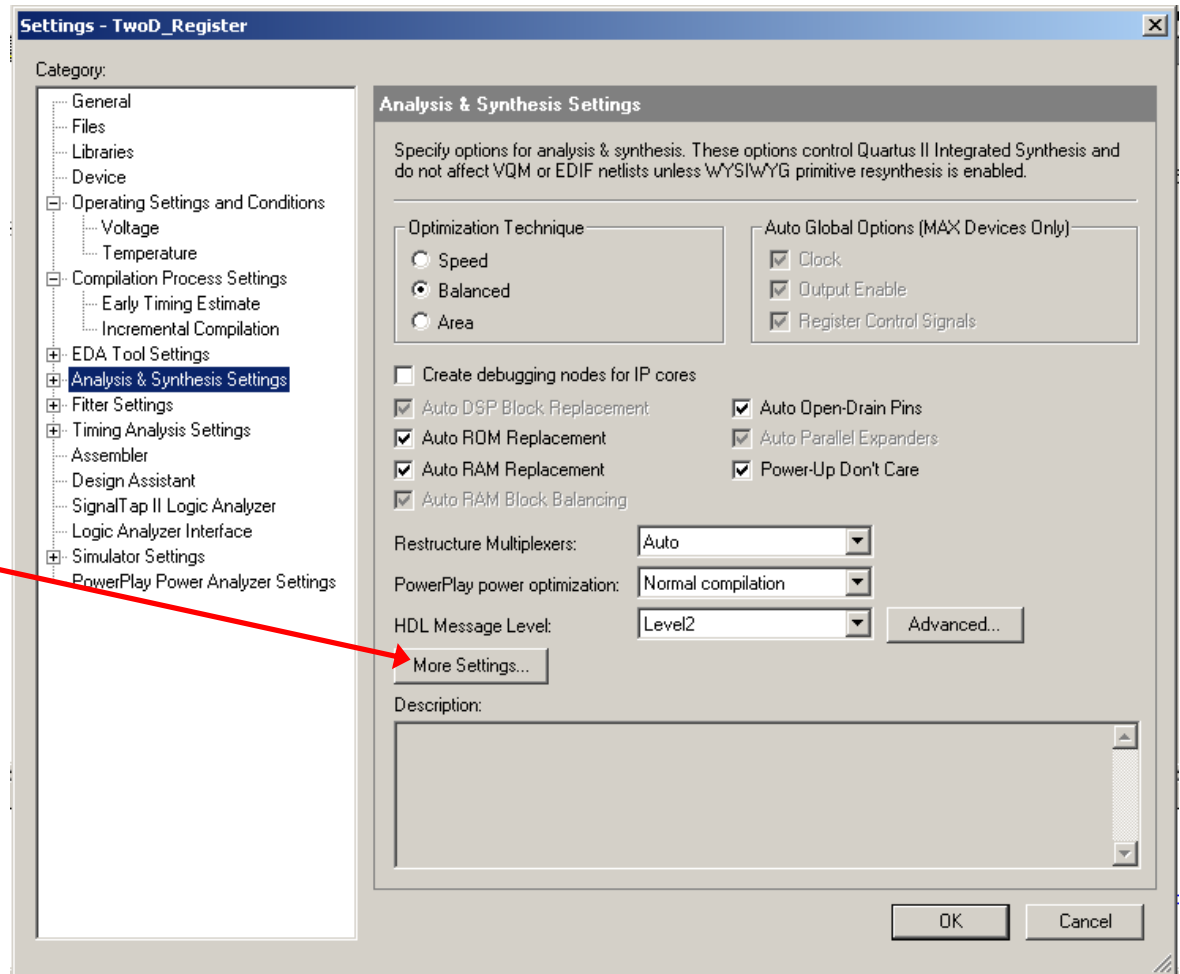
# Logic Element or RAM memory ?

If your memory space is not very large, the compiler will automatically choose to implement your Verilog circuit with D-type flip-flops of the Logic Elements.

You can force the compiler to use the dedicated FPGA memory:

Assignments > Settings >

Click here



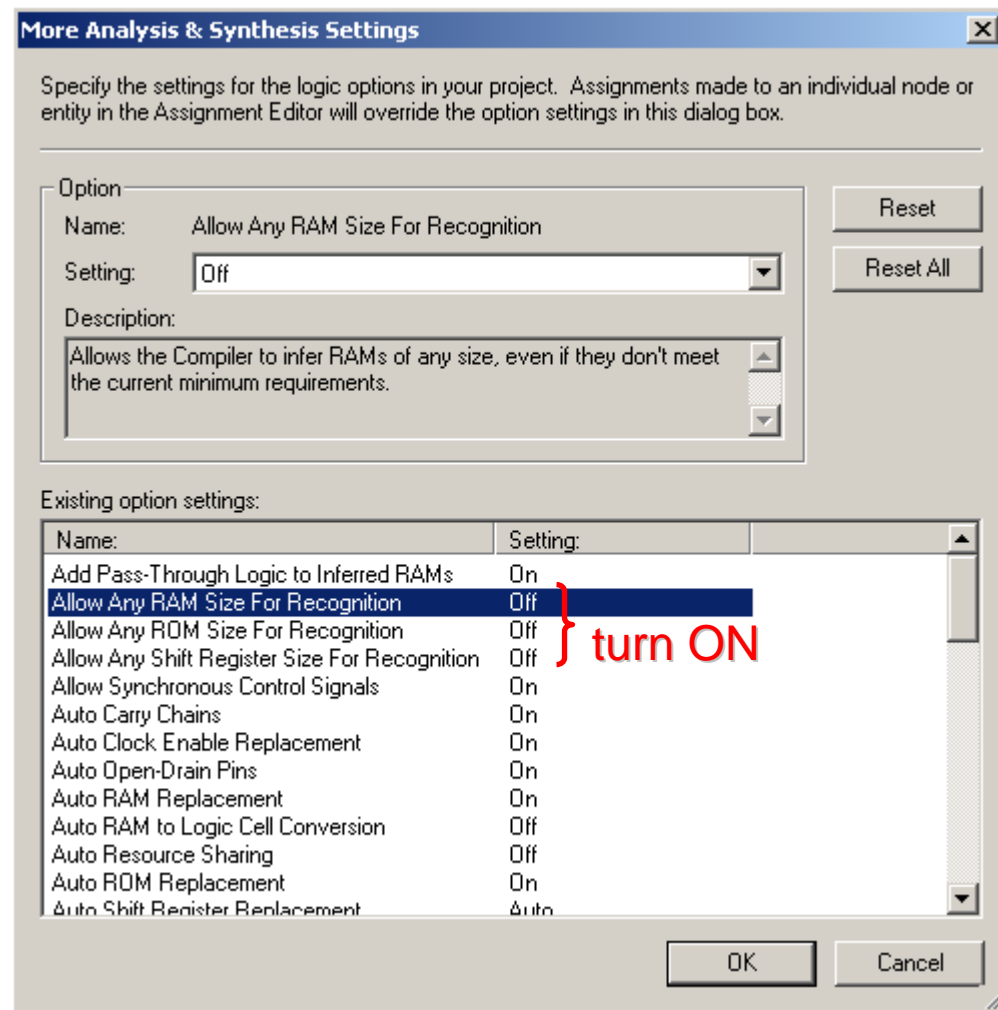


# Logic Element or RAM memory ?

If your memory space is not very large, the compiler will automatically choose to implement your Verilog circuit with D-type flip-flops of the Logic Elements.

You can force the compiler to use the dedicated FPGA memory:

Assignments > Settings >



# Dedicated Memory Usage

Check compiler report for memory usage:

Flow Status	Successful - Mon Oct 22 03:03:49 2007
Quartus II Version	7.1 Build 156 04/30/2007 SJ Web Edition
Revision Name	TwoD_Register
Top-level Entity Name	TwoD_Register
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	N/A
Total logic elements	55
Total combinational functions	55
Dedicated logic registers	31
Total registers	31
Total pins	9
Total virtual pins	0
Total memory bits	256
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Dedicated Memory Usage

