

# FPGA Tutorial on DE1 Board

## TUTORIAL 3

Using Quartus II V13.0

Marsi 2014 - University of TRIESTE

# Tutorial FPGA n.3

---

## *Realizzazione di un generatore di segnali sinusoidali su DE1*

**Descrizione:** La DE1 monta un DECODER [1,2] a frequenze audio, programmabile tramite il protocollo I<sup>2</sup>C [3,4] . Si deve inizialmente realizzare un circuito atto alla configurazione del decoder e successivamente, utilizzando funzioni precostituite si realizzi un generatore di segnali sinusoidali [5-7] e lo si interfacci al precedente DECODER.

**Scopo:** Approfondimento del linguaggio Verilog, gestione e controllo di protocolli di comunicazione, utilizzo di blocchi funzionali costruiti da terze parti, utilizzo di strumenti per il debugging fisico di un sistema operante su FPGA.

### **Apprendimenti previsto:**

- Sviluppo di un sistema di comunicazione I2C
- Testing “on board” tramite “Signal Tap II \_ Logic Analyzer”
- Utilizzo delle “mega core functions”
- Conversione Parallelo-seriale

### **Procedimento:**

Si inizi un nuovo progetto per Ciclone II - EP2C20F484C7N

## *Realizzazione di un generatore di segnali I2C*

Si realizzi un blocco atto a generare i segnali I<sup>2</sup>C [1-3]. In ingresso al blocco oltre al clock ed al reset (sul fronte negativo) vi sia il dato da trasmettere (a 16 bit) un linea “valid” atta a confermare la congruità del segnale in ingresso ed una linea “ack” che viene interrogata ogni nove bit di trasmissione per verificare se il dispositivo ricevente abbia o meno ricevuto il messaggio di “ricevuto” ovvero “acknowledge” (detta linea verrà successivamente collegata con la linea SDA. In uscita siano presenti oltre i segnali SDA ed SCL propri del protocollo I<sup>2</sup>C un segnale “data\_rdy” che segnala che il dispositivo è pronto a ricevere nuovi valori in ingresso ed una linea di “error” che sarà attivata qualora il ricevitore non accusi “ricevuto” alla fine della trasmissione del messaggio.

Una realizzazione del modulo potrebbe essere la seguente:

```
module gen_i2c (clk_in, res, datain, ack_in, valid, sda, scl, data_rdy, error);  
  
    input    clk_in;  
    input    res;  
    input    [15:0] datain;  
    input    valid;  
    input    ack_in;  
    output   reg sda=1;  
    output   reg scl=1;  
    output   reg data_rdy;  
    output   reg error;  
  
    reg [8:0] counter;  
    reg [3:0] cmd_sda;  
    reg [3:0] cmd_scl;  
    reg ACK1,ACK2,ACK3;  
  
    wire [1:0] counter_4=counter[1:0];  
    wire [6:0] counter_bit=counter[8:2];  
    wire [23:0] bits={8'h34,datain};
```

```

// Trans_ON segnale per indicare il periodo di trasmissione dati attiva
always @(posedge clk_in or negedge res) begin
if (!res)
    data_rdy=1;
else if (valid)
    data_rdy=0;
else if (counter_bit ==7'd33)
    data_rdy=1;
end

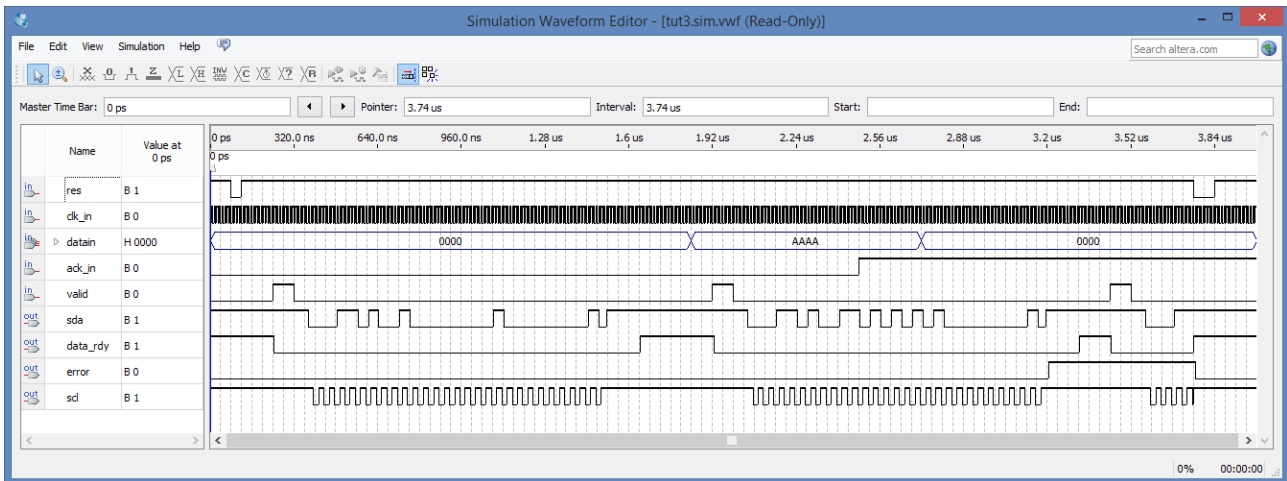
// generazione dei segnali
always @(posedge clk_in or negedge res or posedge valid) begin
if (!res|valid)
begin
counter=0;sda=1;scl=1;
end
else if (!data_rdy)
begin
counter=counter+1;
case (counter_bit)
6'd0 : begin cmd_sda=4'b1111; cmd_scl=4'b1111; end
// start
6'd1 : begin cmd_sda=4'b0001; cmd_scl=4'b0111; end
// device
6'd2 : begin cmd_sda={4{bits[23]}}; cmd_scl=4'b0110; end
6'd3 : begin cmd_sda={4{bits[22]}}; cmd_scl=4'b0110; end
6'd4 : begin cmd_sda={4{bits[21]}}; cmd_scl=4'b0110; end
6'd5 : begin cmd_sda={4{bits[20]}}; cmd_scl=4'b0110; end
6'd6 : begin cmd_sda={4{bits[19]}}; cmd_scl=4'b0110; end
6'd7 : begin cmd_sda={4{bits[18]}}; cmd_scl=4'b0110; end
6'd8 : begin cmd_sda={4{bits[17]}}; cmd_scl=4'b0110; end
6'd9 : begin cmd_sda={4{bits[16]}}; cmd_scl=4'b0110; end
// ACK1
6'd10 : begin cmd_sda=4'b1111; cmd_scl=4'b0110; end
// address
6'd11 : begin cmd_sda={4{bits[15]}}; cmd_scl=4'b0110; end
6'd12 : begin cmd_sda={4{bits[14]}}; cmd_scl=4'b0110; end
6'd13 : begin cmd_sda={4{bits[13]}}; cmd_scl=4'b0110; end
6'd14 : begin cmd_sda={4{bits[12]}}; cmd_scl=4'b0110; end
6'd15 : begin cmd_sda={4{bits[11]}}; cmd_scl=4'b0110; end
6'd16 : begin cmd_sda={4{bits[10]}}; cmd_scl=4'b0110; end
6'd17 : begin cmd_sda={4{bits[9]}}; cmd_scl=4'b0110; end
6'd18 : begin cmd_sda={4{bits[8]}}; cmd_scl=4'b0110; end
// ACK2
6'd19 : begin cmd_sda=4'b1111; cmd_scl=4'b0110; end
// data
6'd20 : begin cmd_sda={4{bits[7]}}; cmd_scl=4'b0110; end
6'd21 : begin cmd_sda={4{bits[6]}}; cmd_scl=4'b0110; end
6'd22 : begin cmd_sda={4{bits[5]}}; cmd_scl=4'b0110; end
6'd23 : begin cmd_sda={4{bits[4]}}; cmd_scl=4'b0110; end
6'd24 : begin cmd_sda={4{bits[3]}}; cmd_scl=4'b0110; end
6'd25 : begin cmd_sda={4{bits[2]}}; cmd_scl=4'b0110; end
6'd26 : begin cmd_sda={4{bits[1]}}; cmd_scl=4'b0110; end
6'd27 : begin cmd_sda={4{bits[0]}}; cmd_scl=4'b0110; end
// ACK3
6'd28 : begin cmd_sda=4'b1111; cmd_scl=4'b0110; end
// stop
6'd29 : begin cmd_sda=4'b1000; cmd_scl=4'b1110; end
// end
6'd30 : begin cmd_sda=4'b1111; cmd_scl=4'b1111; end
endcase
sda=cmd_sda[counter_4];
scl=cmd_scl[counter_4];
end
end

// verifica ACK
always @(posedge clk_in)
if (!res) begin ACK1=1;ACK2=1;ACK3=1;error=0;end
else
begin
if (counter_bit==6'd10 & counter_4==3) ACK1=ack_in;
if (counter_bit==6'd19 & counter_4==3) ACK2=ack_in;
if (counter_bit==6'd28 & counter_4==3) ACK3=ack_in;
if (counter_bit > 6'd29) error=ACK1|ACK2|ACK3;
end
endmodule

```

Forse il file proposto non risulta propriamente “elegante” da un punto di vista strettamente estetico e potrebbe essere riadattato in modo da essere più compatto e forse anche più leggibile. Va peraltro sottolineato che non sempre (anzi, quasi mai) una stesura elegante del codice dal punto di vista della “compattezza” viene poi sintetizzata in modo efficace a livello circuitale.

- Si setti temporaneamente il file come “top Level Entity”
- Si generi un opportuno file di stimoli
- Si simuli funzionalmente il precedente modulo e se ne verifichi il corretto funzionamento.



Si noti ad esempio:

- Che la trasmissione avviene qualche istante dopo che il segnale “valid” ritorna basso
- Che il dato in ingresso non deve essere modificato se prima la linea *data\_rdy* non ritorna allo stato alto (altrimenti il messaggio trasmesso perde di congruità)
- Che i primi 8 bit competono all’indirizzo del dispositivo (h34) mentre gli altri 16 sono inerenti al dato da trasmettere
- La presenza di errore (alla fine della trasmissione) se la linea *ack\_in* non è bassa in presenza del nono bit trasmesso di ogni parola.
- Che in fase di simulazione, NON disponendo di un dispositivo reale capace di interagire con la linea SDA in tri-state, ci si debba “inventare” la presenza di un ulteriore segnale *ack\_in* (al momento pilotato dall’utente, ma che successivamente sarà collegato all’ingresso della linea SDA) per rilevare ipotetici errori di trasmissione.

### Realizzazione di un blocco che fornisca la sequenza I2C

Si realizzi ora un blocco che interagendo coi segnali del blocco sovraesposto fornisca con la corretta sequenza i segnali da far giungere al DECODER. Il sistema riceve in ingresso oltre il *clock* ed il *reset* il segnale *data\_rdy* con il quale rileva che il dispositivo ricevente è pronto a ricevere i dati. In uscita fornisce il dato da trasmettere ed il segnale di attivazione “*valid*”. Una soluzione potrebbe essere la seguente:

```

`define rom_size 6'd11

module gen_code (clk,res,rdy_in,data,valid );
    input  clk,res,rdy_in;
    output reg [15:0] data;
    output reg valid;

```

```

        reg [2:0] fasi;
        reg [3:0] addr;
        reg [15:0]ROM[`rom_size:0];

// contatore delle fasi
always @(posedge clk or negedge res) begin
    if (!res) fasi<=3'b000;
    else if (!rdy_in) fasi<=3'b000;
    else if ((rdy_in)&(fasi < 3'b111)) fasi<=fasi+3'b001;
    end

// generazione segnale "valid"
always @(posedge clk or negedge res) begin
    if (!res) valid=0;
    else if ((fasi==3'b101)&(addr < `rom_size)) valid=1;
    else valid=0;
end

// incremento dell'indirizzo
always @(posedge clk or negedge res) begin
    if (!res) addr = 0;
    else if ((addr < `rom_size)&(fasi==3'b001)) addr=addr+1; end

//dato in uscita
always @(posedge clk)
begin
    ROM[0]= 16'h0000;           // dummy
    ROM[1]= 16'h001A;           // linvol
    ROM[2]= 16'h021A;           // rinvol
    ROM[3]= 16'h047B;           // lhpvol
    ROM[4]= 16'h067B;           // rhpvol
    ROM[5]= 16'h0828;           // audiopath (h0810)
    ROM[6]= 16'h0a06;           // digitalpath - deenfasi
    ROM[7]= 16'h0c00;           // power down disable
    ROM[8]= 16'h0e41;           // format and master 0e02
    ROM[9]= 16'h1000;           // control
    ROM[10]= 16'h1201;          // active
    ROM[11]= 16'h0000;          // dummy

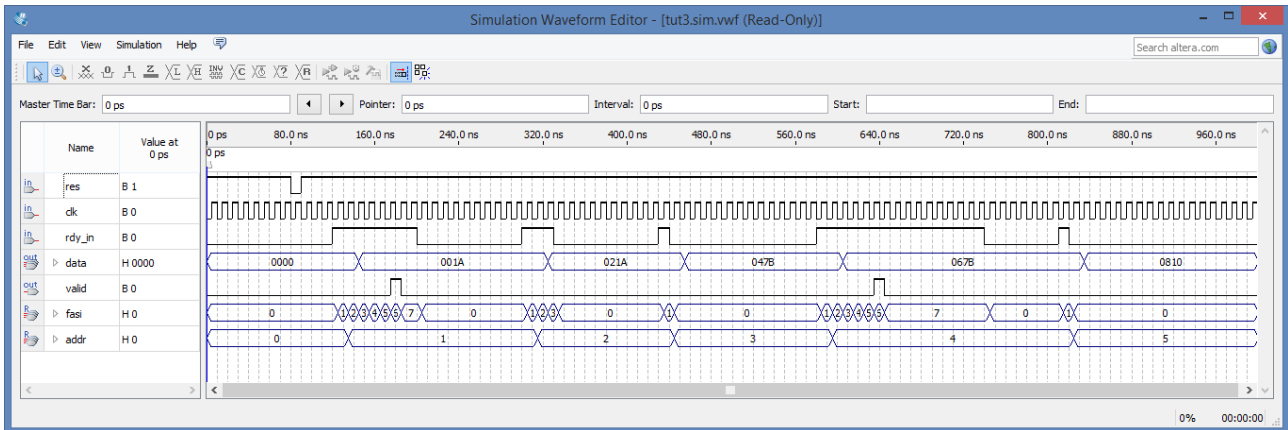
    data=ROM[addr];
end
endmodule

```

Da notare in particolare che tutti i segnali sono sincronizzati sul medesimo clock onde prevenire problemi di disallineamento del clock. Inoltre per scandire le varie fasi della trasmissione si è creato un “contatore delle fasi” che viene impiegato per temporizzare le varie fasi della trasmissione.

- Esso effettua il conteggio solo quando “data\_rdy” è attivo
- E’ un contatore a saturazione che non supera mai il valore 7
- All’istante 001 incrementa si incrementa l’indirizzo di memoria
- All’istante 101 Viene alzato il segnale “valid” che si riabbasserà all’istante successivo

Si verifichi tramite opportuna simulazione funzionale il corretto funzionamento del modulo.



Si noti in particolare

- Che in pratica il dato all'indirizzo "0" non può essere mai letto, infatti quando arriva primo fronte su "data\_rdy" il dato cambia subito dopo. Pertanto all'indirizzo 0 verrà riempito con un dato "dummy".
- Che il segnale "valid" viene generato solo se il segnale "data\_rdy" rimane alto un tempo sufficiente.
- Viceversa l'incremento di indirizzo avviene anche se "data\_rdy" rimane allo stato alto per un tempo inferiore (ciò è legato alla temporizzazione e al contatore delle fasi). A questo problema si farà fronte con una opportuna comunicazione tra i blocchi.

Nota: per accedere a segnali che non siano solamente quelli relativi alle porte di I/O del modulo, all'interno della finestra "node Finder" si faccia riferimento ai segnali disponibili quando si predispone il filtro a: *Design Entry (all names)*. Questo fermo restando che il simulatore integrato all'interno di Quartus esegue una simulazione a livello RTL, motivo per il quale il circuito che si simula NON è esattamente quello da noi descritto (sebbene il funzionamento sia il medesimo) pertanto può facilmente avvenire che taluni segnali non esistano nel modello RTL. Se si vuole eseguire una simulazione più approfondita e magari anche un debugging del sorgente Verilog, si consiglia di utilizzare il simulatore ModelSim e si rimanda il lettore al tutorial dedicato.

## Realizzazione del sistema completo di configurazione I2C

Poiché il protocollo I2C prevede una frequenza di trasmissione di qualche decina di KHz bisogna predisporre ancora un blocco atto a ridurre opportunamente la frequenza di clock. Una possibile realizzazione è la seguente:

```
module clk_div(clk_in,res,clk_i2c);

input clk_in,res;
output reg clk_i2c;

reg [12:0] div_i2c;

//      Clock Setting

parameter      CLK_Freq      =      50000000;      //      50      MHz
parameter      I2C_Freq      =      20000;         //      20      KHz

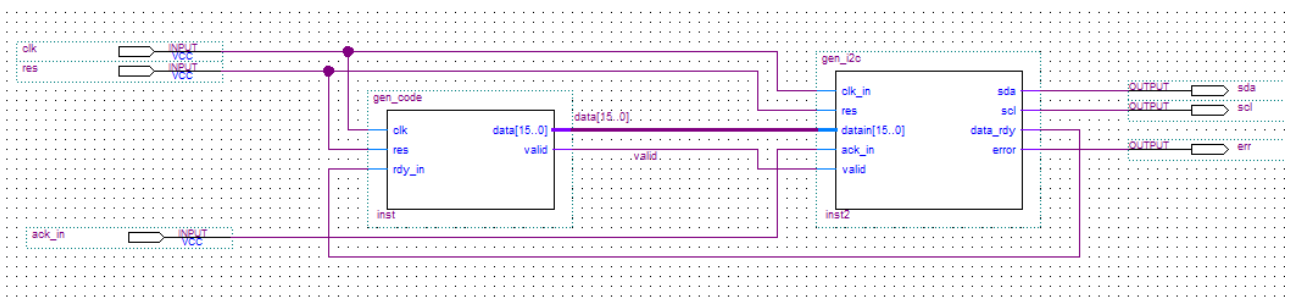
////////// I2c clk //////////
always@(posedge clk_in or negedge res)
begin
    if(!res)
    begin
        div_i2c<=      0;
        clk_i2c<=      0;
    end
    else
    begin
        if( div_i2c < ((CLK_Freq/(I2C_Freq*2))-1) )
        div_i2c<=      div_i2c+1;
        else
        begin
            div_i2c<=      0;
            clk_i2c<=      ~clk_i2c;
        end
    end
end

end

endmodule
```

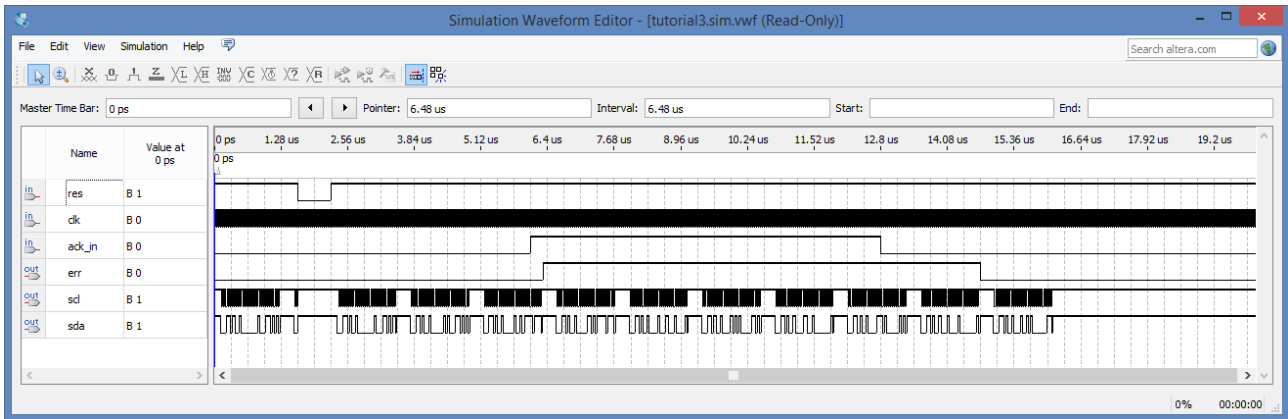
Per i tre moduli così realizzati si costruisca il simbolo corrispondente.

Si noti che ove si desidera simulare lo schematico completo è meglio soprassedere all'impiego del riduttore di frequenza, ed altresì la linea `ack_in` deve essere pilotata manualmente dall'utente onde emulare la presunta risposta del decoder. Notare inoltre l'impiego del segnale `data_rdy` in loop



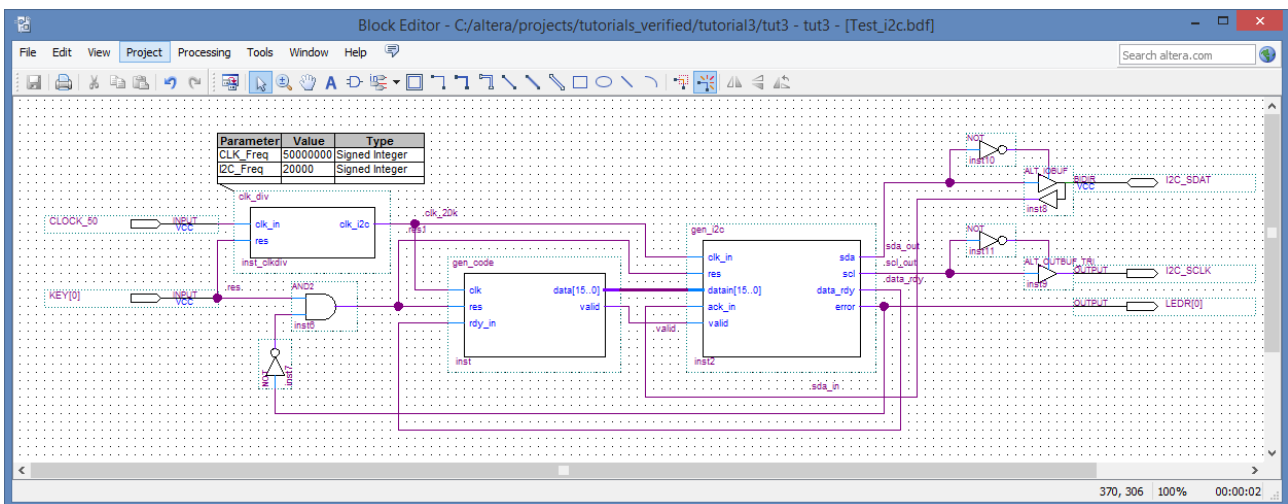
Generando un opportuno file di stimoli si può verificare tramite simulazione funzionale il corretto funzionamento:





Lo schema completo deve peraltro evidenziare oltre ai collegamenti tra i vari blocchi, le porte di controllo e le corrispondenti uscite, inoltre i nomi di questi devono rispecchiare i nomi del file di vincoli. Si tenga peraltro in particolare conto il fatto che i segnali d'uscita SDA ed SCL [3,4] possono assumere o lo stato basso (0) oppure lo stato di alta impedenza (Z), che poi si trasformerà in un valore logico alto grazie alla resistenza di "pullup" integrata sulla scheda DE1. Inoltre si noti che la porta per SDA deve essere di tipo bidirezionale. Per completare il segnale di errore può essere utilizzato, dopo essere stato invertito per "resettare" il sistema.

Lo schema completo potrebbe essere il seguente:




Questo sistema non può essere simulato poiché mancante del segnale di ACK in ingresso che dovrebbe pervenire dalla linea che si collega al decoder. Si può però verificarne il corretto funzionamento direttamente sulla scheda.

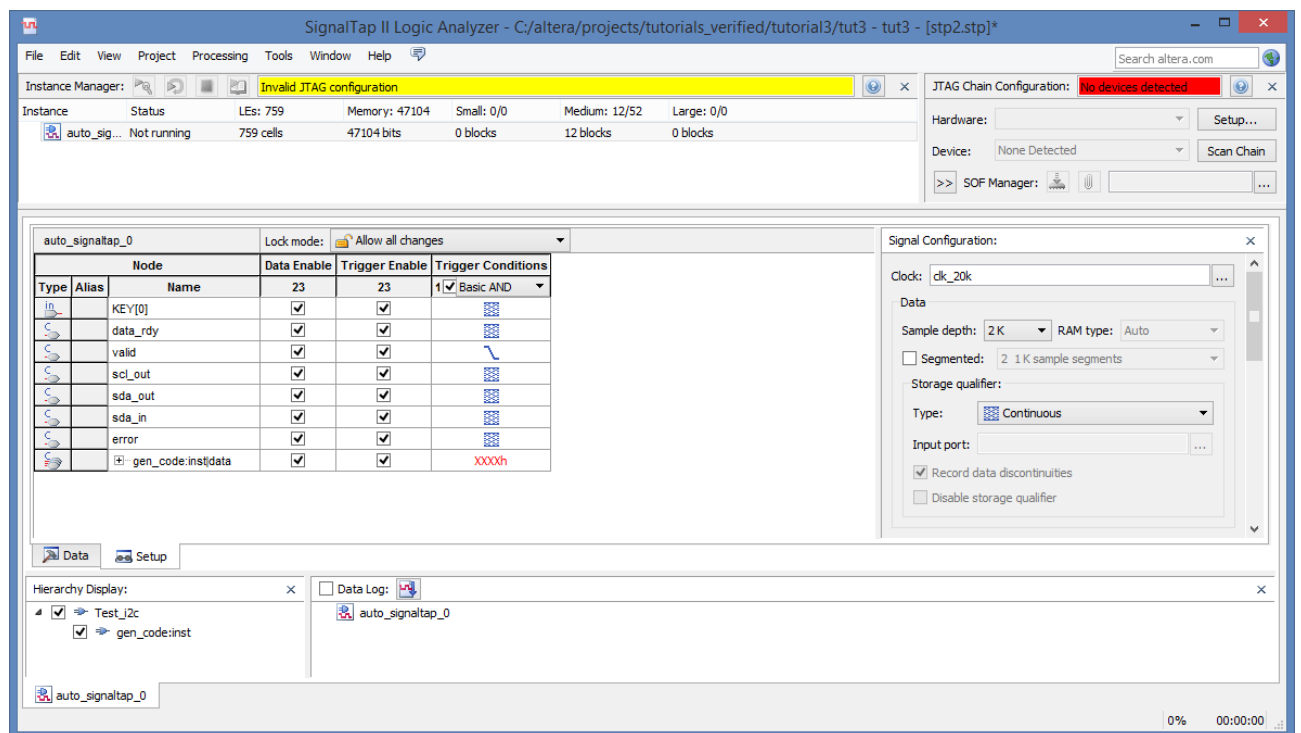
### Testing su Scheda

- Si effettui una compilazione parziale del progetto
  - Si definisca l'attuale blocco quale Top-Level-Entity
  - Start Analysis and Synthesis
- Si aggiunga al progetto un nuovo file di tipo Signal Tap II – Logic Analyzer [8,9]
  - File > New – Signal Tap II – Logic Analyzer
- Si definisca, nella sezione relativa a "signal configuration", inizialmente un clock sul quale sincronizzare l'acquisizione dei dati (si consiglia di usare il clock a bassa frequenza (ovvero quello in



uscita dal riduttore di frequenza) piuttosto che il clock a 50 MHz che eseguirebbe un campionamento troppo fitto dei segnali da analizzare. Per fare ciò è consigliabile nominare opportunamente la corrispondente “net” sullo schematico (clk\_20k).

- Cliccare su “...” nella scheda Signal Configuration – per quanto concerne il segnale di clock
- Far eventualmente comparire le opzioni attraverso l’apposito tasto 
- Filtrare tramite “Signal Tap II – Pre Synthesis”
- Scegliere il clock (clk\_20k)
- Si definisca una profondità di memoria (Sample depth) di 2K
  - Sample depth = 2K
- Si definisca quali segnali monitorare (per identificarli comodamente risulta comodo renderli univoci dando essi un nome all’interno dello schematico)
  - Edit > Add Node
  - Filtrare tramite l’opzione “Signal Tap II – Pre Synthesis”
  - Scegliere i segnali da monitorare
- Si definisca su quale trigger sincronizzare l’acquisizione (ad esempio si può utilizzare il fronte negativo del segnale valid)
  - Right click on Trigger condition
  - Scegliere l’opzione desiderata

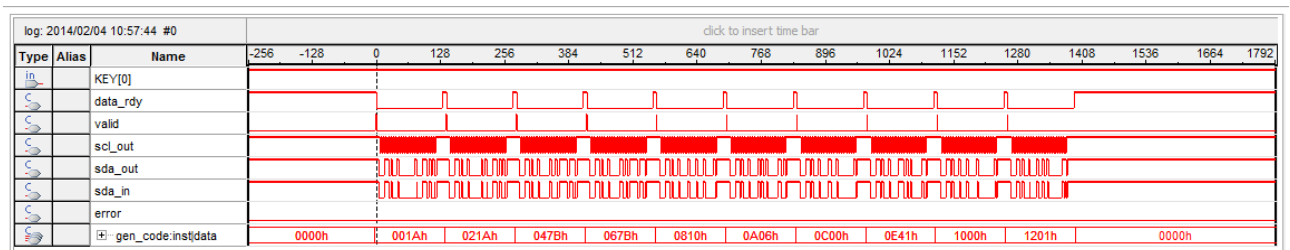


- Si salvi il file così realizzato.
- Si esegua una compilazione completa del progetto – ricordandosi di imporre i vincoli corretti ai piedini di I/O.
- Qualora Si utilizzi la versione “Quartus Web” si deve attivare (una tantum) l’opzione TalkBack [10]: per fare questo bisogna andare nel direttorio dove sono presenti gli eseguibili di Quartus

C:\altera\13.0sp1\quartus\bin  
e lanciare

tb2\_install.exe

- Si effettui il download su scheda
- Nella Finestra JTAG Chain Configuration verificare che l'Hardware coincida con l'USB Blaster e che il dispositivo rilevato coincida con l'FPGA montata sulla DE1.
- Si inizi l'analisi dei segnali
  - Processing > Run Analysis (F5)
  - L'analisi inizia ma il sistema attende che si verifichi la condizione di trigger stabilita per iniziare l'acquisizione dei segnali
- Si attivi il tasto preposto al reset sulla scheda (KEY[0])
- Si verifichi la correttezza delle forme d'onda in uscita



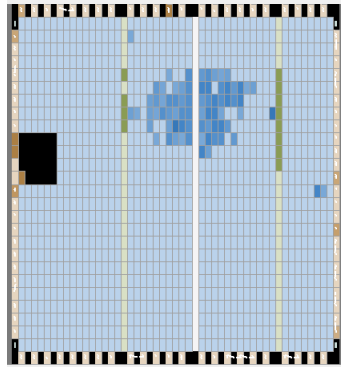
Si noti in particolare come la forma d'onda rilevata in ingresso su SDA (sda\_in) coincida con quella in uscita (sda\_out), fatta eccezione per ogni nono bit, momento nel quale il decoder posto sulla scheda segnala l'ACK abbassando detta linea durante tutto il periodo per il quale SCL rimane alto.

Si provi ora a modificare l'indirizzo del dispositivo e si verifichi che quest'ultimo, non riconoscendo la trasmissione come a lui diretta, non segnali alcun ACK! Questo crea un segnale d'errore che resetta continuamente il sistema ritentando la trasmissione all'infinito. Vedere le f.d.o generate. Questo viene reso evidente anche dal fatto che il segnale di errore è collegato al LEDR[0] che pertanto continua ad accendersi e spegnersi ad una frequenza superiore a quelle che il sistema visivo umano possono discriminare, apparendo pertanto debolmente illuminato.

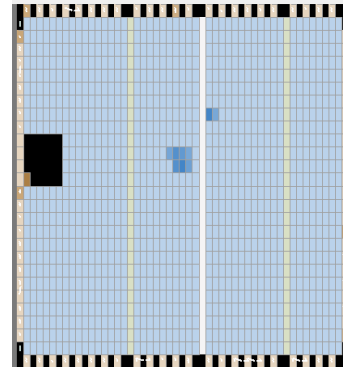
Si noti che l'impiego del Signal Tap Analyzer richiede molte risorse Hardware al progetto, pertanto una volta verificato il corretto funzionamento si possono liberare le risorse. Per fare ciò

Assignments > Settings

Nella scheda relativa al Signal Tap II Logic Analyzer, disattivare l'opzione "Enable Signal Tap II Logic Analyzer". Si consiglia di visualizzare le risorse utilizzate (attraverso reports e Chip Planner) tanto quando l'opzione è attivata che quando non lo è. In particolare si noti che l'impiego di questo strumento deve giocoforza riservare delle considerevoli porzioni di memoria per la raccolta dei dati che poi vengono visualizzati, e per questo scopo ricorre alle memorie integrate M4K. Questo oltre ovviamente a parecchi blocchi logici per gestire l'intero sistema di testabilità.



Con Signal Tap



Senza Signal Tap

### Configurazione del Decoder[1,2]

Facendo riferimento agli 11 registri del decoder, provare a configurarlo secondo varie metodologie.

REGISTER	B 15	B 14	B 13	B 12	B 11	B 10	B 9	B8	B7	B6	B5	B4	B3	B2	B1	B0
R0 (00h)	0	0	0	0	0	0	0	LRIN BOTH	LIN MUTE	0	0	LINVOL				
R1 (02h)	0	0	0	0	0	0	1	RLIN BOTH	RIN MUTE	0	0	RINVOL				
R2 (04h)	0	0	0	0	0	1	0	LRHP BOTH	LZCEN	LHPVOL						
R3 (06h)	0	0	0	0	0	1	1	RLHP BOTH	RZCEN	RHPVOL						
R4 (08h)	0	0	0	0	1	0	0	0	SIDEATT	SDETONE	DAC SEL	BY PASS	INSEL	MUTE MIC	MIC BOOST	
R5 (0Ah)	0	0	0	0	1	0	1	0	0	0	0	HPOR	DAC MU	DEEMPH	ADC HPD	
R6 (0Ch)	0	0	0	0	1	1	0	0	PWR OFF	CLK OUTPD	OSCPD	OUTPD	DACPD	ADCPD	MICPD	LINEINPD
R7 (0Eh)	0	0	0	0	1	1	1	0	BCLK INV	MS	LR SWAP	LRP	IWL		FORMAT	
R8 (10h)	0	0	0	1	0	0	0	0	CLKO DIV2	CLKI DIV2	SR				BOSR	USB/NORM
R9 (12h)	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	ACTIVE
R15(1Eh)	0	0	0	1	1	1	1	RESET								
	ADDRESS							DATA								

- Collegare una sorgente sonora in ingresso ed un sistema per l'ascolto (casse-cuffia) in uscita
- Provare a modificare il volume in ingresso e/o in uscita
- Provare a modificare la sorgente attivando solo il convertitore DAC– escludere sidetone, bypass e configurare linein per la linea in ingresso (h0810)
- Introdurre un blocco che ricevendo in ingresso un clock a 50 MHz ne generi uno a 12,5 MHz e si usi quest'ultimo per pilotare il master clock del decoder. Si può impiegare il medesimo blocco realizzato prima, pur di modificarne i parametri (basta agire esclusivamente a livello di schematico) senza modificare il sorgente verilog.
- Successivamente configurare il decoder come Master (esso genera tutti i segnali) (h0E41)

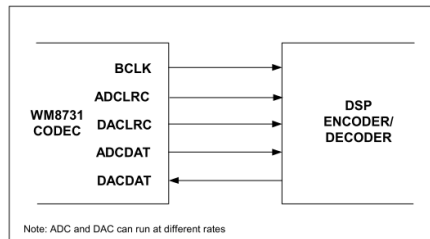
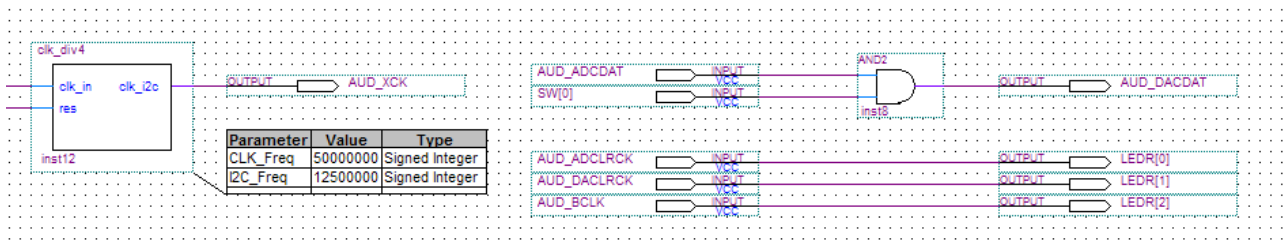
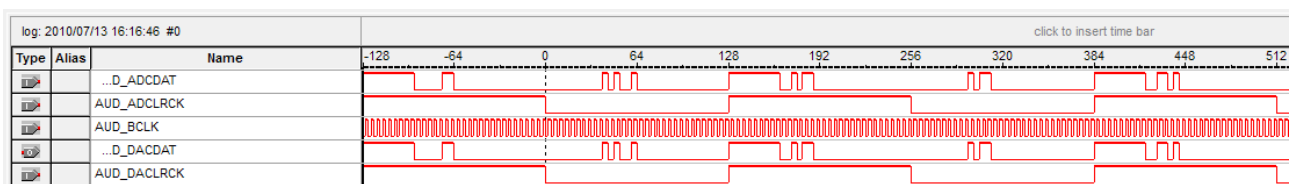


Figure 30 Master Mode

- Si aggiunga quindi allo schematico la seguente configurazione



- Si provi ad utilizzare il “Signal Tap II – Logic Analyzer” per monitorare i segnali audio generati dal decoder, configurando il clock di acquisizione su AUD\_XCK.
- Si analizzino visivamente i segnali riportati al variare del volume del segnale in ingresso. Quali sono i bit interessanti? Dove si trovano i bit più significativi? Perché a volume basso i bit della parola sono quasi tutti uguali (tutti a 0 o tutti a 1)?



### Generazione di un segnale sinusoidale

Si utilizzino le “megafunction” [7] per la realizzazione di un sistema in grado di generare un segnale digitale sinusoidale. Queste realizzano una serie molto variegato di blocchi sviluppati da terze parti. Nella fattispecie noi relizzeremo un NCO (Numerically Controlled Oscillator)

- Tool > Mega Wizard Pulgin Manager
- Create a New custom Megafunction Variation
- Si scelga DSP > Signal Generator > NCO v13.0
- Si assegni un nome opportuno, il linguaggio di implementazione (Verilog HDL) e la famiglia di FPGA (Cyclone II)
- Nella parametrizzazione del modulo (Step 1):
  - Nella Cartella Parameters
    - Si fissi a 16 bit di precisione a tutti e tre i segnali: Phase accumulator precision, Angular resolution, Magnitude Precision.
    - Disattivare : implement phase dithering

- Si scelga la tipologia di algoritmo (Cordic) [5,6] da implementare e si visualizzino le risorse impiegate (nell'opportuna sotto-cartella "Resource Estimate")
  - NOTA: la frequenza di clock in ingresso e la frequenza desiderata in uscita non modificano l'implementazione del blocco ma aiutano a calcolare il valore da fornire in ingresso al blocco stesso per realizzare la frequenza desiderata.
- Nella cartella implementation si predisponga
  - Un solo canale (single Output)
  - Disattivare: frequency modulation Input
  - Disattivare: Phase Modulation Input
  - Cordic Implementation Parallel
  - Finish
- Nella setup Simulation (Step 2)
  - Attivare Generate Simulation Model
  - Linguaggio: VerilogHDL
  - OK
- Generate (Step 3)
  - A Processo ultimato: Exit
  - alla domanda: Do you want to add IP to the project: rispondere YES

Ora all'interno del progetto vi è il blocco generato che può essere integrato nel nostro progetto. La simulazione di tale blocco a livello di Gate NON è peraltro consentita e non è consentita neppure la simulazione a livello RTL (quindi sono interdette le simulazioni all'interno di Quartus), ma vi è la possibilità di simularlo esclusivamente a livello comportamentale. Per fare ciò bisogna ricorrere ad un simulatore comportamentale esterno, come ad esempio ModelSim

Inoltre per consentire la sintesi completa del progetto bisogna disattivare la richiesta di generazione di files per la simulazione attraverso tools di terze parti:

Assignment > Settings

Nella scheda relativa a "EDA Tools" accertarsi che tutti siano <None>

### *Simulazione con ModelSim*

In fase di generazione del blocco per la realizzazione del generatore sinusoidale, sono stati generati molti altri files dedicati alla simulazione con Modelsim. Tra questi vi sono in particolare

- <file>.vo : descrittore comportamentale del generatore sinusoidale
- <file>\_tb.v : file di stimoli per una simulazione comportamentale
- <file>\_vo\_msim.tcl : serie di comandi in format .tcl da passare al simulatore
- <file>\_wave.do : serie di comandi per una corretta visualizzazione delle forme d'onda

Inoltre vi sono anche alcuni files dedicati alla simulazione attraverso Matlab (che non verranno considerati in questo contesto).

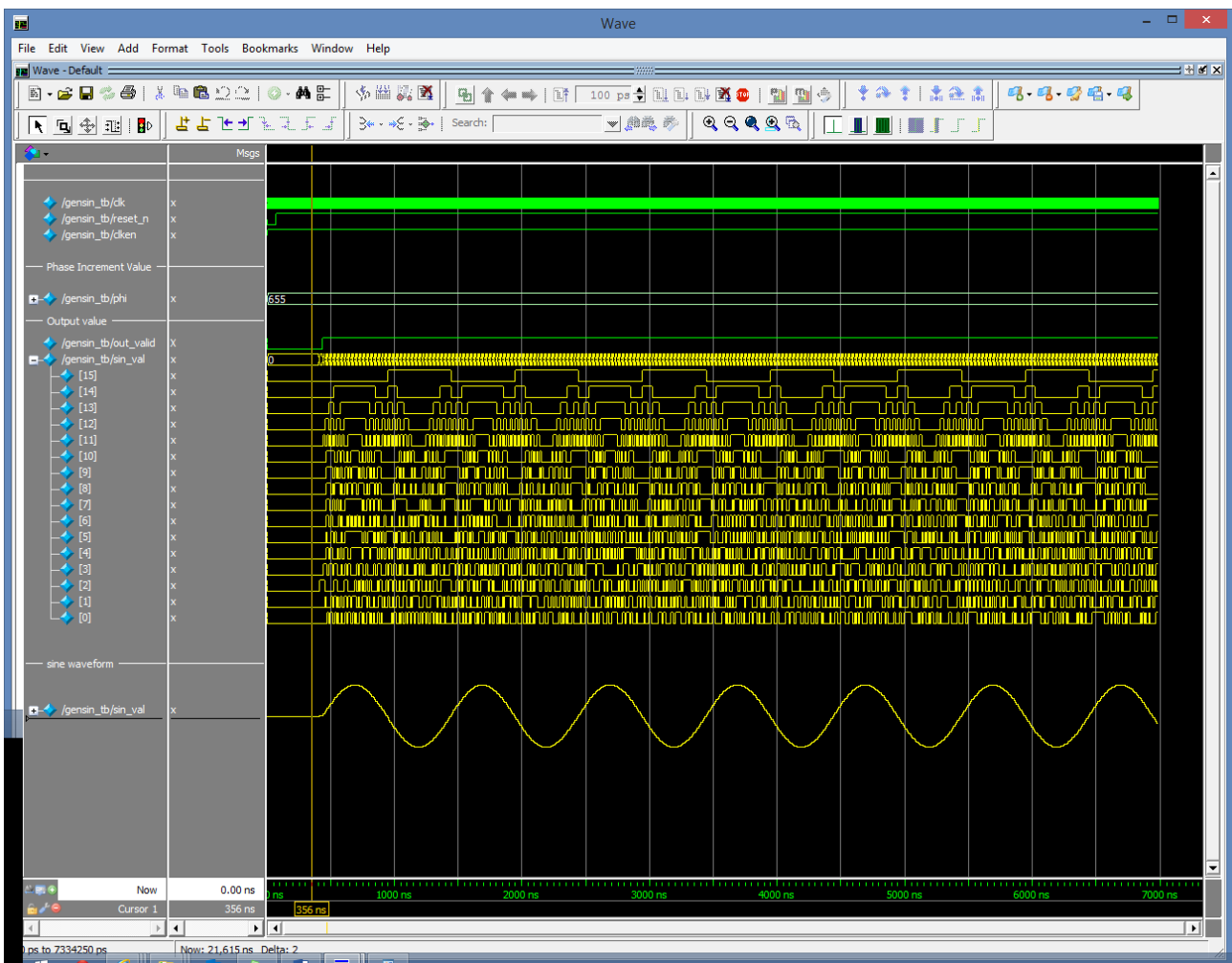
Per attivare la simulazione

1. Lanciare Modelsim
2. Nella finestra di console spostarsi nel direttorio dove sono localizzati i files da simulare attraverso il comando "cd: ..."
3. Nella finestra di console attivare lo script .tcl attraverso il comando  

```
Do <nomefile>_vo_msim.tcl
```

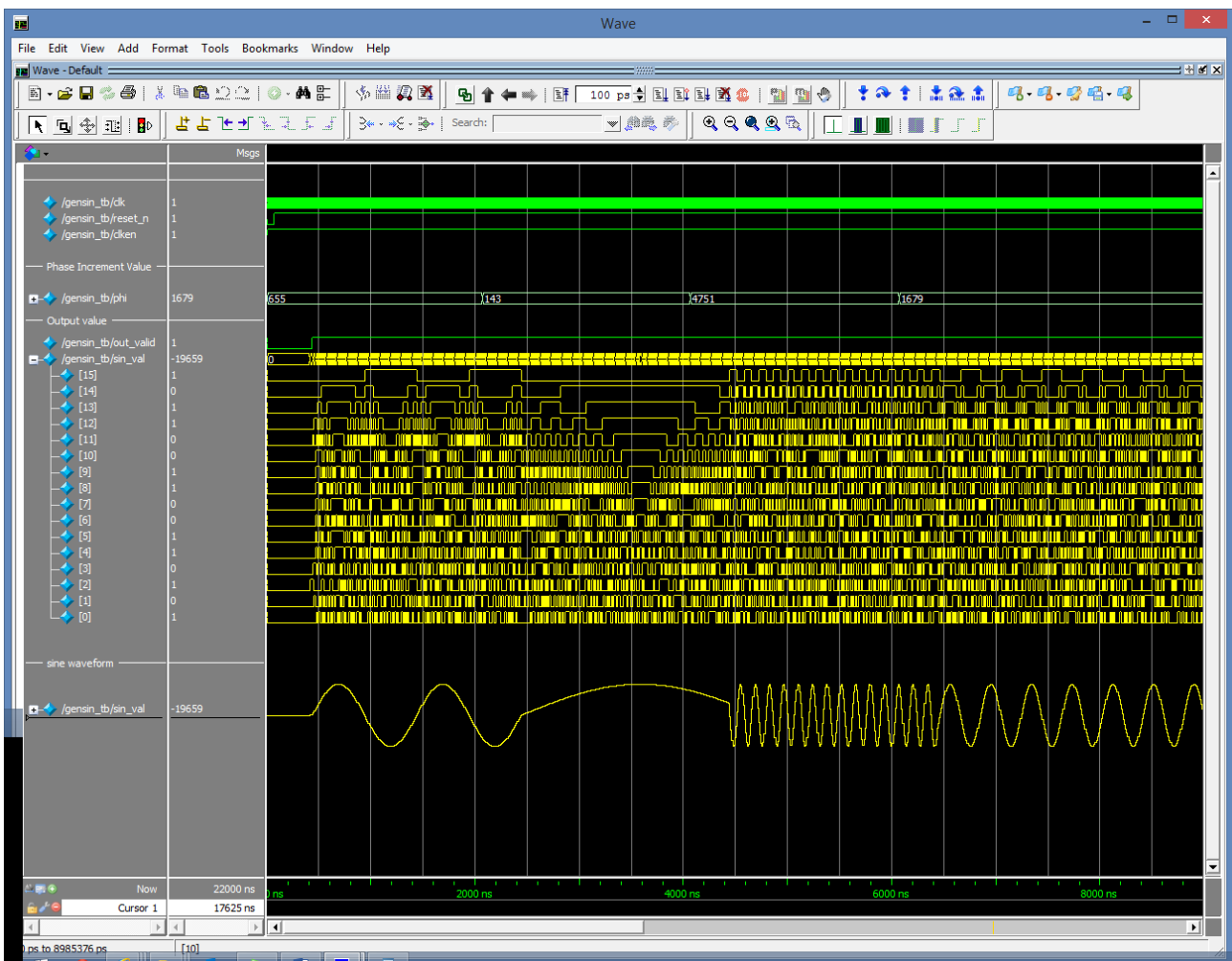
Tale script esegue in sequenza:
  1. Genera un file di log
  2. Crea un nuovo Progetto (o apre quello già esistente)
  3. Crea la Libreria di Lavoro (o apre quella già esistente)
  4. Compila le varie librerie su cui sia appoggiata
  5. Compila il file di descrizione comportamentale
  6. Compila il file di stimoli
  7. Fa partire il simulatore linkando le opportune librerie
  8. Apre la finestra per la visualizzazione delle f.d.o, include i segnali da visualizzare e la formatta opportunamente (tramite il file <nomefile>\_wave.do)
  9. Attiva un run di simulazione della durata di 22000 ns

Il risultato dopo un po' di tempo viene visualizzato



Eventualmente si può leggermente ritoccare il file di stimoli per provare a generare diverse frequenze in diversi istanti di tempo:

```
begin
  $dumpvars;
  #0 clk = 1'b0;
  #0 reset_n = 1'b0;
  #0 clken = 1'b1;
  #0 phi = 16'b0000001010001111;
  #(14*HALF_CYCLE) reset_n = 1'b1;
  #(200*CYCLE) phi = 16'b0000000010001111;
  #(200*CYCLE) phi = 16'b0001001010001111;
  #(200*CYCLE) phi = 16'b0000011010001111;
end
```



Per un approfondimento sull'uso di ModelSim si rimanda al tutorial dedicato.

*Realizzazione di un blocco per la generazione dei segnali audio*



Per completare il progetto si deve ancora realizzare un blocco che avendo in ingresso i dati del generatore di sinusoidi (oltre ovviamente al clock a 50 Mhz ed al reset) generi in uscita tutti i segnali per pilotare il decoder.

In particolare questo dovrà generare:

- Il master clock del decoder a circa 12.288 Mhz (nel nostro caso saranno 12.500 MHz ma una certa tolleranza è consentita)
  - $50 \text{ MHz} / 2^2$
- Il clock di alternanza L/R (a 48.8 KHz)
  - $50 \text{ MHz} / 2^{10}$
- Il Bit clock – supponendo di predisporre il sistema per gestire 32 bits per canale sarà a circa (3.12 MHz)
  - $50 \text{ MHz} / 2^4$
- Il flusso di dati audio seriali

Una possibile soluzione è qui di seguito rappresentata:

```

module gen_aud(d,clk_in,res,clk_mck,clk_bck,clk_lrck,bit_out);

input [15:0] d;
input clk_in;
input res;
output reg clk_mck;
output reg clk_bck;
output reg clk_lrck;
output reg bit_out;

reg [9:0] counter;
reg [4:0] bit_counter; // 32 bits

wire cont_mck=counter[0];
wire [2:0] cont_bck=counter[2:0];
wire [3:0] cont_bit=counter[3:0];
wire [8:0] cont_lrck=counter[8:0];
reg [15:0] data;

////////// AUD Generator //////////
always@(posedge clk_in or negedge res)
begin
    if(!res)
        begin
            counter<= 8'b00000000;
            clk_mck <= 0 ; clk_bck <= 0 ;clk_lrck <= 0 ;
            bit_counter <= 4'b0000;
        end
    else
        begin
            counter <= counter + 8'b00000001;
            // MCLK (12.5 MHz)
            if(cont_mck==1) clk_mck <= ~clk_mck;
            // BCLK (3.12 MHz) predisposto per 32 bit-canale
            if(cont_bck==7) clk_bck <= ~clk_bck;
            // LRCLK (48.8 KHz)
            if(cont_lrck==511)
                begin
                    data[15:0]<={d[0],d[1],d[2],d[3],d[4],d[5],d[6],d[7],d[8],d[9],d[10],d[11],d[12],d[13],d[14],d[15]};
                    clk_lrck <= ~clk_lrck;
                    bit_counter <= 0;
                end
        end
end

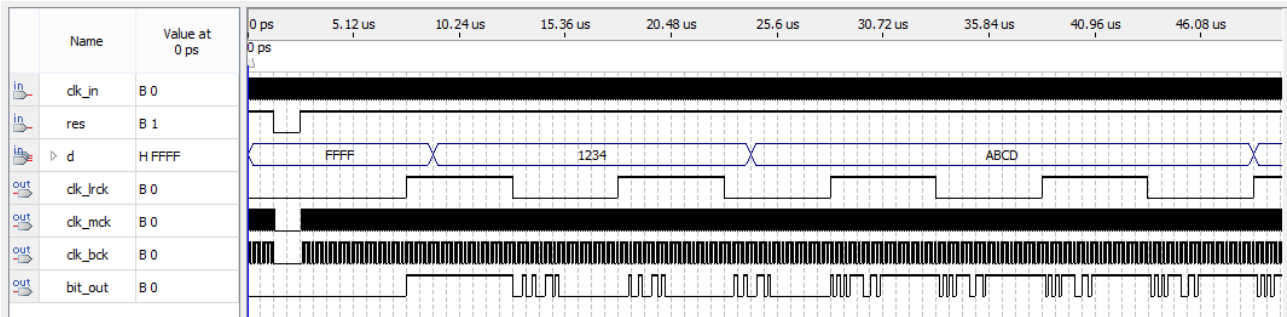
```

```

        end
        // bit counter
        if(cont_bit==0 & bit_counter <= 15)
            begin
                bit_counter <= bit_counter+1;
                bit_out <= data[bit_counter];
            end
        end
    end
endmodule

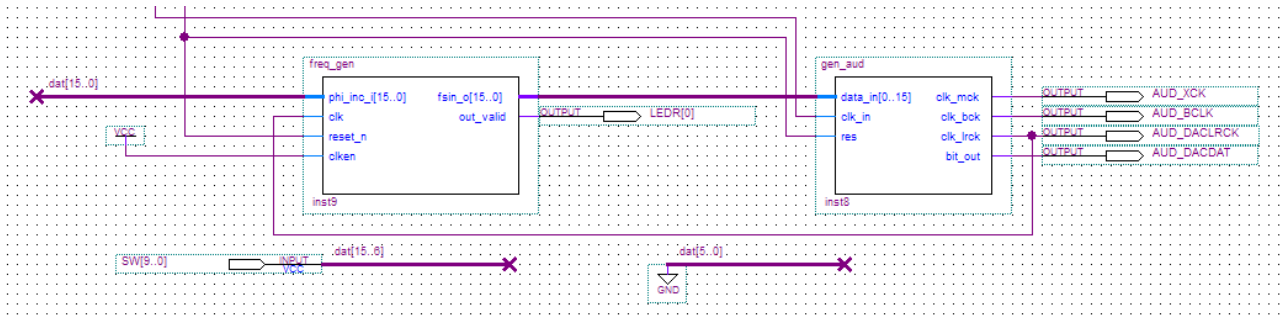
```

Che può essere opportunamente simulato per verificarne il corretto funzionamento



Si noti che la stringa seriale dei 16 bit componenti il messaggio da inviare risultano giustificati a sinistra col bit più significativo a sinistra e che tutti i bit oltre il 16imo (inutilizzati) replicano l'ultimo.

Si generi il simbolo corrispondente e successivamente si realizzi nello schematico (a livello Top Level) i seguenti collegamenti:



Si noti che mentre gli swithes pilotano i dieci bit più significativi del segnale di "fase\_incrementale" (ovvero di pulsazione) mentre i 6 bit meno significativi sono forzati al valore 0.

Si noti ancora l'uso del clock di alternanza (LR) come clock di campionamento (per determinare la successione dei vari campioni)

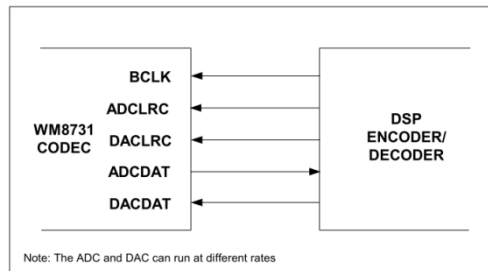
L'uscita data\_valid è portata sul LEDR [0]

Si riconfiguri il decoder in modo tale che funzioni da Slave ed utilizzi il formato "left justified" a 16 bits (h0E01).

Ci si ricordi, prima della compilazione di verificare che non sia richiesta da parte di Quartus l'uso di tool da terze parti

Assigments > Settings (ctrl-shift-E)

Nella scheda "EDA Tools Settings" tutte le opzioni siano <none>



Si compili l'intero sistema e si effettui il download su DE1. Ci si assicuri di usare il file corretto per il download su scheda (che prende un nome diverso da quello usato finora) avendo questo l'estensione "time\_limited"

Si noti in particolare che poiché il progetto include un elemento (il generatore sinusoidale) il cui uso è concesso solo dietro licenza, peraltro assente nella versione "web", il sistema funziona a tempo indeterminato solo fintanto che il collegamento USB è attivato ed il relativo controllo è attivo. Nell'eventualità il collegamento USB venga staccato (nel qual caso bisogna ovviamente predisporre un'alimentazione alternativa), il sistema funziona solo per un tempo limitato.

Nota:

Nel'uso del Signal Tap II – Logic Analyzer, potrebbe essere comodo, ove si possieda la licenza, attivare il modo di funzionamento a "compilazione incrementale" [17] che consentirebbe di risparmiare tempo in fase di compilazione. Capita pertanto talvolta che questo modo si attivi automaticamente, se in particolare non si possiede la licenza per questa opzione, la compilazione risulta impossibile. Per disattivare questa funzione si deve: digitare, nella finestra TCL il seguente comando:

Aprire la console tcl e digitare:

```
View > Utility Windows > TCL console (Alt-2)
"set_global_assignment -name INCREMENTAL_COMPILATION OFF"
```

## Bibliografia

- [1] Wolfson Microelectronics: **WM8731 / WM8731L**  
“**Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates**”  
<http://www.cs.columbia.edu/~sedwards/classes/2008/4840/Wolfson-WM8731-audio-CODEC.pdf>
- [2] Wolfson Microelectronics: “**Simplified Datasheet for WM8731 Audio Codec**”  
[http://diglab.ece.gatech.edu/downloads/project/WM8731\\_SimplifiedDatasheet.pdf](http://diglab.ece.gatech.edu/downloads/project/WM8731_SimplifiedDatasheet.pdf)
- [3] Philips Semiconductors: “**The I2C-bus and how to use it (including specifications)**”  
[http://www.i2c-bus.org/fileadmin/ftp/i2c\\_bus\\_specification\\_1995.pdf](http://www.i2c-bus.org/fileadmin/ftp/i2c_bus_specification_1995.pdf)
- [4] Philips Semiconductors: “**The I2C-bus specification**”  
<http://www.cs.unc.edu/Research/stc/FAQs/Interfaces/I2C-BusSpec-V2.1.pdf>
- [5] Derek Nowrouzezahrai Brian Decker William Bishop: “**Efficient Double-Precision Cosine Generation**”  
[http://www.iro.umontreal.ca/~derek/files/cosine\\_report.pdf](http://www.iro.umontreal.ca/~derek/files/cosine_report.pdf)
- [6] Altera Corporation “**CORDIC Reference Design**”  
<http://www.altera.com.cn/literature/an/an263.pdf>
- [7] Altera Corporation “**NCO MegaCore Function – User Guide**”  
[http://www.altera.com/literature/ug/ug\\_nco.pdf](http://www.altera.com/literature/ug/ug_nco.pdf)
- [8] Altera Corporation “**Design Debugging Using the SignalTap II Logic Analyzer**”  
[http://www.altera.com/literature/hb/qts/qts\\_qii53009.pdf](http://www.altera.com/literature/hb/qts/qts_qii53009.pdf)
- [9] Altera Corporation “**SignalTap II with Verilog Designs**”  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/10.1/Tutorials/Verilog/SignalTap.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/10.1/Tutorials/Verilog/SignalTap.pdf)
- [10] Altera –Support Solution “**Solution ID: rd06202008\_168**”  
[http://www.altera.com/support/kdb/solutions/rd06202008\\_168.html](http://www.altera.com/support/kdb/solutions/rd06202008_168.html)
- [11] Alter Corporation: “**Using ModelSim to Simulate Logic Circuits for Altera FPGA Devices**”  
[ftp://ftp.altera.com/up/pub/Altera\\_Material/10.1/Tutorials/Using\\_ModelSim.pdf](ftp://ftp.altera.com/up/pub/Altera_Material/10.1/Tutorials/Using_ModelSim.pdf)
- [12] Mentor Graphics: “**ModelSim Reference Manual**”  
[http://cseweb.ucsd.edu/classes/fa10/cse140L/lab/docs/modelsim\\_ref.pdf](http://cseweb.ucsd.edu/classes/fa10/cse140L/lab/docs/modelsim_ref.pdf)
- [13] Mentor Graphics: “**ModelSim User Manual**”  
[http://homepages.cae.wisc.edu/~ece554/new\\_website/ToolDoc/Modelsim\\_docs/docs/pdf/se\\_man.pdf](http://homepages.cae.wisc.edu/~ece554/new_website/ToolDoc/Modelsim_docs/docs/pdf/se_man.pdf)
- [14] Mentor Graphics “**ModelSim Tutorial**”  
<http://www.fatih.edu.tr/~onur/download/EEE546/modelsimTutorial.PDF>
- [15] Larbi Boughaleb “**ModelSim Tutorial**”  
[http://www.ece.northwestern.edu/~ismail/courses/c92/ModelSim\\_Tutorial.pdf](http://www.ece.northwestern.edu/~ismail/courses/c92/ModelSim_Tutorial.pdf)
- [16] “**ModelSim Tutorial**”  
<http://ee.hawaii.edu/~sasaki/EE361/Fall06/Lab/Lab4.1/ModelSim.pdf>
- [17] Altera Corporation: “**Quartus II Incremental Compilation for Hierarchical and Team-Based Design**”  
[http://www.altera.com/literature/hb/qts/qts\\_qii51015.pdf](http://www.altera.com/literature/hb/qts/qts_qii51015.pdf)

