

UART, SPI e PWM SU FPGA

Nell'evoluzione dell'attuale progettazione, l'integrazione in minimi spazi e costi contenuti è sempre un argomento grande interesse.

L'articolo vuole evidenziare la possibilità di integrare periferiche aggiuntive, per un microprocessore di scheda, su un'economica e compatta FPGA della nuova famiglia Nano A3PN, fornita da ACTEL

Frequentemente, durante lo sviluppo di un progetto, vengono a mancare alcune periferiche semplici, ma importanti, poiché quelle rese disponibili dal microprocessore scelto sono già utilizzate per altre funzioni. In tale situazione, anziché ricercare subito un modello di microprocessore fornito di periferiche aggiuntive (e spesso può capitare che non lo si trovi nella famiglia attualmente in uso, costringendoci a cambi non indolori), è utile valutare l'integrazione delle periferiche mancanti su FPGA, che per il loro costo competitivo, permettono di raggiungere l'obiettivo senza penalizzare l'aspetto finanziario del costo scheda. L'esempio che presentiamo permette l'integrazione di una seriale UART, di una seriale sincrona Master di tipo SPI e di tre canali PWM a 8 bit, il tutto in un chip da 30K gate e con un costo minimo. Si tenga sempre presente che, nel caso di sviluppi eseguiti su FPGA, uno dei vantaggi im-

portanti è che l'eventuale integrazione successiva di altre periferiche è sempre possibile, semplicemente migrando al modello di FPGA successivo (nel nostro caso alla 60K gate), con un aumento di costo contenuto, ma senza rifare nulla a livello di PCB. Il progetto può essere provato sia in simulazione, utilizzando i tools di sviluppo gratuiti messi a disposizione da ACTEL, in quanto fornitore del silicio, sia in HW utilizzando l'AGLN-NANO-KIT come strumento di debug HW. Nel caso foste interessati al test HW è necessario procurarsi un qualsiasi schedino di sviluppo per microprocessori, per la programmazione delle periferiche integrate nella FPGA. L'intero progetto e la relativa documentazione (manuali delle macro utilizzate) è scaricabile dai riferimenti riportati a fine articolo, mentre l'ambiente di sviluppo completo **LIBERO 8.5** è disponibile gratuitamente sul sito del produttore www.actel.com

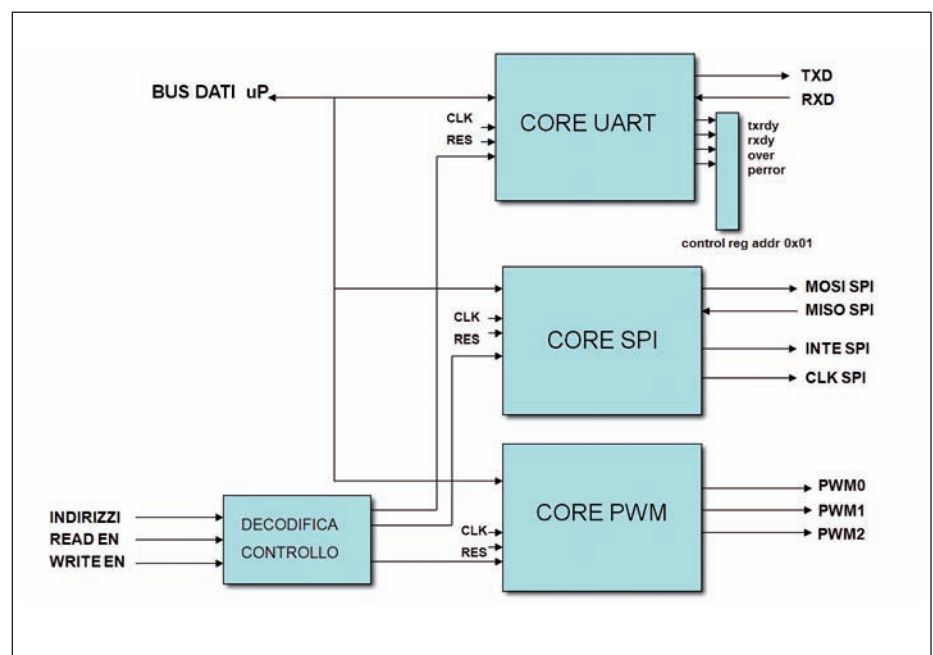


Figura 1:

Struttura del progetto

Con l'ambiente Libero 8.5 regolarmente installato, se avete scaricato il nostro progetto, è sufficiente, dopo aver lanciato Libero 8.5, selezionare menu Project -> Open project e con un doppio click su file di progetto PRJ aprirlo (**figura 0**). Si noti che essendo già compilato correttamente, le icone di sintesi e Place and Route sono colorate in verde. Il progetto come schema a blocchi è presentato in **figura 1** ed è composto fondamentalmente da tre blocchi: il Core UART, il core SPI e il core PWM. Nel caso del Core SPI e UART abbiamo due moduli, che vengono forniti gratuitamente da ACTEL in formato offuscato: si intende, quindi, che dei due moduli abbiamo a disposizione il manuale, esattamente come se fosse un chip, ma non ci è permesso la sua modifica. Il modulo PWM, per la sua semplicità, è invece presente in codice

sorgente. Tutte e tre i moduli hanno un'interfaccia a bus parallelo 8 bit, che verrà utilizzata dal microcontrollore (esterno in questo caso), per accedere alle singole periferiche.

Il clock di base deve essere fornito dall'esterno, mentre il Reset HW previsto, sempre dall'esterno, opera a livello attivo basso.

Top level del progetto: Prova_spi_actel.vhd

La **figura 2** ci introduce alla parte codice sorgente del progetto. Il top del progetto è prova_spi_actel.vhd. In questo file abbiamo dichiarato i tre moduli periferici SPI, UART e PWM.

Per la selezione e la gestione delle periferiche UART e SPI, vediamo che sono stati decodificati alcuni segnali fra cui gli indirizzi ADDR 0-4, per cui avremo che la SPI risponderà all'indirizzo 0x8, mentre la UART avrà come indirizzo base 0x0.

Per la gestione delle singole periferiche la funzione Writereg consente la programmazione di alcuni registri, creati come interfaccia fra il micro esterno e le periferiche usate.

In particolare reg_count_pwmX conterrà il valore di duty cycle desiderato per ognuno dei singoli PWM, baud_reg, come si intuisce dal nome, definirà la velocità di trasmissione della UART, regENABPWM usa i primi tre bit per l'abilitazione dei relativi canali PWM, infine, regprescaler permette di definire la frequenza dei canali PWM. Nel progetto, qui semplificato, la frequenza è comune per tutte e tre i canali e come facile espansione i lettori potranno creare tre prescaler separati, agguinando alla funzione Writereg gli opportuni registri utili alla loro programmazione e creando due nuovi gruppi prescaler (**figura 3**) e freecounter. Qui il processo genera il segnale Tc, che è di fatto

Listato 1

```
WR_SPI <= '1' when (ADDR(3 downto 2) = "10" and WR = '1') else '0';
RD_SPI <= '1' when (ADDR(3 downto 2) = "10" and RD = '1') else '0';
WR_UART <= '0' when (ADDR(3 downto 0) = "0000" and WR = '1') else '1';
RD_UART <= '0' when (ADDR(3 downto 0) = "0000" and RD = '1') else '1';
CS_UART <= '0' when (ADDR(3 downto 0) = "0000" and (RD = '1' or WR = '1')) else '1';

writeReg:
  process (RESN, CLK, WR) begin
    if (RESN='0') then
      baud_reg <= "01101000"; -- 68h per clock 16 mhz circa 9540 baud
      regENABPWM <= (others => '0');
      regprescaler <= "00001111";
    elsif (CLK'event and CLK='0') then
      if ((WR='1') and (ADDR(3 downto 0)="0001")) then --pwm0
        reg_count_pwm0 <= DATA;
      else if ((WR='1') and (ADDR(3 downto 0)="0010")) then --pwm1
        reg_count_pwm1 <= DATA;
      else if ((WR='1') and (ADDR(3 downto 0)="0011")) then --Pwm2
        reg_count_pwm2 <= DATA;
      else if ((WR='1') and (ADDR(3 downto 0)="0100")) then --baud reg
        baud_reg <= DATA;
      else if ((WR='1') and (ADDR(3 downto 0)="0101")) then --enam pwm ADD5
        regENABPWM <= DATA;
      else if ((WR='1') and (ADDR="0111")) then -- prescale pwm clock add 6
        regprescaler <= DATA;
      end if;
    end if;
  end if;
end if;
end process writeReg;
```

l'end count del prescaler, che viene passato al contatore libero, il quale sulla cadenza del segnale Tc stesso incrementerà il contatore freecount (8 bit).

Generazione PWM: PWMchannel.vhd

La **figura 4** ci riporta la sezione di codice del semplice comparatore utilizzato in PWMchannel.

Il comparatore, come si vede, utilizza RunningCounter, che non è altro che il freecounter passato dal top level, mentre StopValue sarà il contenuto del registro reg_count_pwmX scritto, sempre nel top level. Il comparatore al raggiungimento della soglia provvede a mettere a zero il segnale RunningEqualStop.

La funzione PWMprocess permette la generazione del PWM di uscita, controllando lo stato dei relativi bit EnablePWM (in arrivo dal top level registro regE-NABPWM) e RunningEqualStop

Seriale SPI: CORE SPI

Il blocco CORESPI utilizza una macro di tipo offuscato fornita gratuitamente da ACTEL. Il relativo data sheet (allegato al progetto che avete scaricato) descrive e permette di programmare il blocco esattamente come se avessimo utilizzato una periferica sviluppata in silicio. La **figura 5** ci evidenzia i suoi registri di programmazione, che sono disponibili al micro esterno. Partendo dal registro ad indirizzo 0x0, troviamo il registro dati in trasmissione e ricezione. Il registro control register 1 a indirizzo 0x01 ci permette di abilitare o meno interrupt di sistema, programmare la modalità master o slave, definire l'ordine di uscita dei bit serializzati (MSB o LSB) e del clock di trasmissione e ricezione ed, infine, ci consente di programmare fase, polarità e valore. Nel control register 2, all'indirizzo 0x02, troviamo i bit per l'abilitazione della SPI e per

la cancellazione di errori in ricezione/trasmissione. Sempre all'indirizzo 0x02, ma in lettura, troviamo lo status register, che ci permette di controllare se la seriale SPI è abilitata, se siamo in trasmissione o meno (solo in modalità master), se il registro di trasmissione è vuoto e se in ricezione abbiamo un carattere disponibile per la lettura. Possiamo verificare, infine, se si è verificato un over run, per cui abbiamo ricevuto un nuovo carattere senza aver letto il precedente. Come ultimo registro, all'indirizzo 0x11, disponiamo del registro selezione slave, che è valido solo quando si opera in master mode e ci permette di selezionare fino a otto slave (e relativi otto pin di uscita disponibili). Utile informazione: la macro SPI da sola utilizza, in modalità master, 265 tile o macrocelle della FPGA; pertanto, nel caso della piccola A3PN030, avendone a disposizione 768, la macro occuperà circa un terzo della logica disponibile.

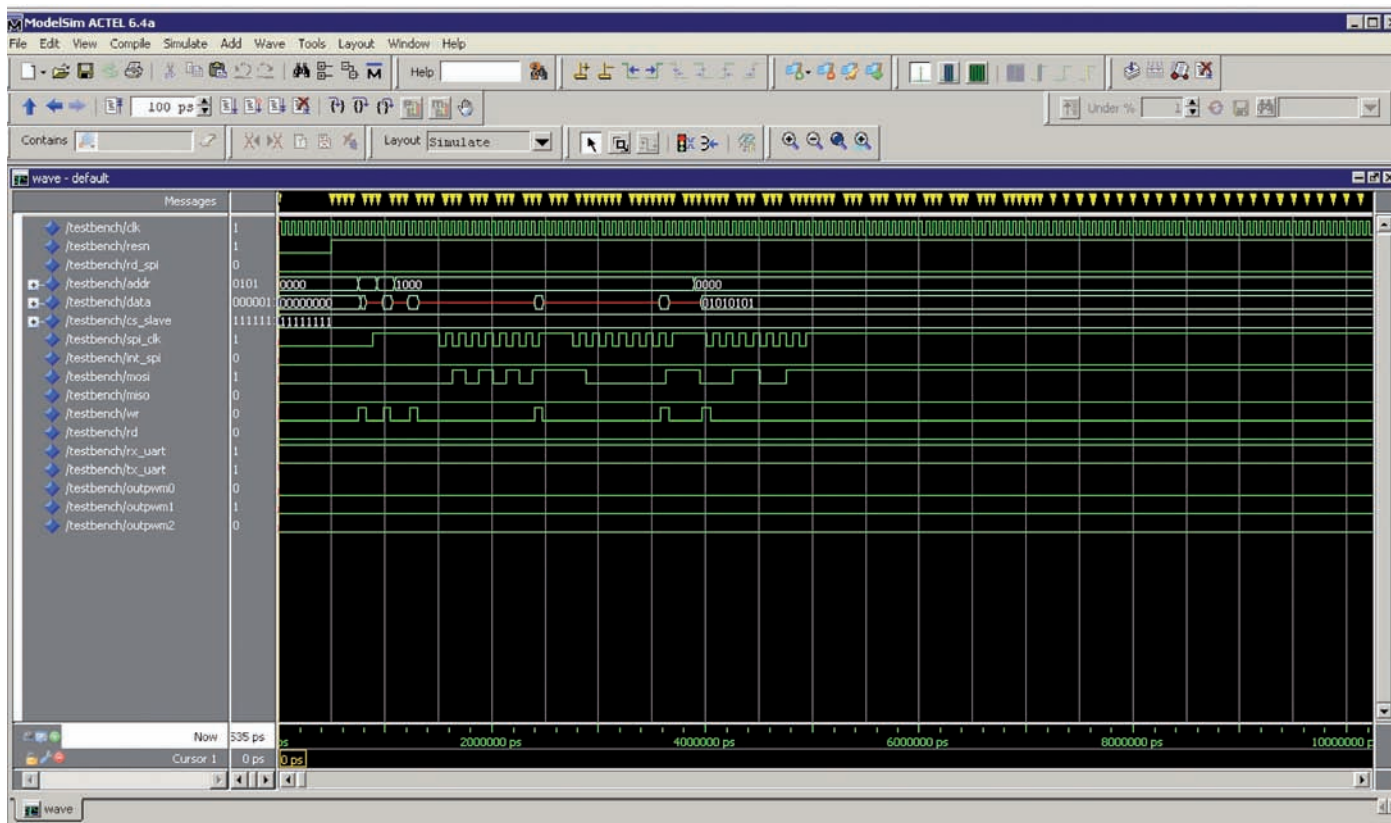


Figura 6:

addrbus[1:0]	Read/Write	Description								
00	read/write	SPI Data Register. Read receive data register/Write transmit data register								
01	read/write	<p>Control Register #1</p> <p>bit 7: Interrupt Enable '1' = Enable '0' = Disable</p> <p>bit 6: Master / Slave mode select '1' = Master '0' = Slave</p> <p>bit 5: Order of data transfer '1' = LSB first '0' = MSB first</p> <p>bit 4: CPHA (Clock Phase)</p> <p>bit 3: CPOL (Clock Polarity)</p> <p>bit 2-0: Serial Clock Rate Select (for Master mode only)</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 50%;">000: $f_{SYSCLK} \div 2$</td> <td style="width: 50%;">100: $f_{SYSCLK} \div 32$</td> </tr> <tr> <td>001: $f_{SYSCLK} \div 4$</td> <td>101: $f_{SYSCLK} \div 64$</td> </tr> <tr> <td>010: $f_{SYSCLK} \div 8$</td> <td>110: $f_{SYSCLK} \div 128$</td> </tr> <tr> <td>011: $f_{SYSCLK} \div 16$</td> <td>111: $f_{SYSCLK} \div 256$</td> </tr> </table>	000: $f_{SYSCLK} \div 2$	100: $f_{SYSCLK} \div 32$	001: $f_{SYSCLK} \div 4$	101: $f_{SYSCLK} \div 64$	010: $f_{SYSCLK} \div 8$	110: $f_{SYSCLK} \div 128$	011: $f_{SYSCLK} \div 16$	111: $f_{SYSCLK} \div 256$
000: $f_{SYSCLK} \div 2$	100: $f_{SYSCLK} \div 32$									
001: $f_{SYSCLK} \div 4$	101: $f_{SYSCLK} \div 64$									
010: $f_{SYSCLK} \div 8$	110: $f_{SYSCLK} \div 128$									
011: $f_{SYSCLK} \div 16$	111: $f_{SYSCLK} \div 256$									
10	write	<p>Control Register #2</p> <p>bit 7: Enable SPI</p> <p>bit 6-1: Not used</p> <p>bit 0: Clear errors</p>								
10	read	<p>Read Status Register</p> <p>bit 7: SPI is enabled</p> <p>bit 6-4: Not used</p> <p>bit 3: SPI in Master mode, currently transferring data. Used to check if the SPI Master is busy, before disabling it</p> <p>bit 2: TX_register_empty. New data for transmission can be written to the Transmit Data Register.</p> <p>bit 1: RX_data_ready. When this bit = '1', the Receive Data Register must be read before the next character is received.</p> <p>bit 0: '1' = Error, generated when a character is received before the previous is read from the RX data register.</p>								
11	read/write	<p>Slave Select Register (Used only in Master mode)</p> <p>bit 7: Slave Select line 7; '1' = Enable, '0' = Disable</p> <p>bit 6: Slave Select line 6; '1' = Enable, '0' = Disable</p> <p>bit 5: Slave Select line 5; '1' = Enable, '0' = Disable</p> <p>bit 4: Slave Select line 4; '1' = Enable, '0' = Disable</p> <p>bit 3: Slave Select line 3; '1' = Enable, '0' = Disable</p> <p>bit 2: Slave Select line 2; '1' = Enable, '0' = Disable</p> <p>bit 1: Slave Select line 1; '1' = Enable, '0' = Disable</p> <p>bit 0: Slave Select line 0; '1' = Enable, '0' = Disable</p>								

Figura 5:

Seriale UART: CORE UART

Il blocco core UART è la seconda macro che utilizziamo, sfruttando sempre la libreria gratuita, che ACTEL mette a disposizione nella suite tools. Analizzando il data sheet di questa IP, vediamo che possiamo decidere di avere o meno delle FIFO sia in ricezione che in trasmissione. Per l'abilitazione delle FIFO è sufficiente passare alla macro il parametro generico TX FIFO e RX FIFO, definendolo a 1 logico. In caso contrario le FIFO rimarranno disabilitate. Per la gestione del Baud rate di ricetrasmisione, la macro ha predisposto un ingresso fisico come registro a 8 bit, che abbiamo opportunamente associato al nostro registro interno baud_reg. Il calcolo del Baud rate può

essere effettuato usando la seguente formula:

Baudrate = $\text{CLK} / (\text{baud_reg} + 1) * 16$, dove CLK sarà il clock di sistema per il nostro progetto. Per fare un esempio con un CLK di 16 Mhz, volendo avere un Baud rate di 9600 bit/sec, dovremo caricare nel registro baud_reg il valore 103. La macro mette, inoltre, a disposizione con uscita a pin le segnalazioni di base per la sua gestione: Txrdy, Rxrdy per il controllo del flusso dati in trasmissione e ricezione, Parity error e Overflow per la verifica degli errori. Questi pin interni sono stati riportati sul registro controlreg accessibile dal microprocessore all'indirizzo 0x01 e con il posizionamento dei bit nella modalità seguente:

bit0 = txrdy; bit1 = rxrdy; bit2= parity_err; bit2= overflow.

Il numero delle celle logiche occupate dal blocco UART è di 226, sempre su un massimo disponibile di 768 per la A3P030N.

IMPLEMENTAZIONE PRATICA

Le prove pratiche su questo progetto possono essere sviluppate avendo a disposizione una schedina di sviluppo di un qualunque microprocessore, che riporti su un connettore tipo strip passo 2,54 mm, se non il bus dati, almeno un paio di porte di I/O, con cui simulare i cicli di bus.

Il progetto è stato in ogni caso provato anche a livello di simulazione e allegato con esso troverete anche la sezione dedicata

RIFERIMENTI

- ■ ■ A3PN Family Hand Book <http://www.actel.com/products/pa3nano/docs.aspx>
- ■ ■ AGLN Nano Kit Reference http://www.actel.com/products/hardware/devkits_boards/igloonano_starter.aspx
- ■ ■ Libero 8.5 suite <http://www.actel.com/download/software/libero/default.aspx>
- ■ ■ Richiesta Progetto completo/ Libero 8.5 su DVD marketing@latecnikadue.com

Listato 2

```

- Processo di scaling frequenza ingresso
process( CLK, RESN) begin
  if( RESN='0') then
    counter <= (others => '0');
    tc <= '0';
  elsif( CLK'event and CLK='1') then
    if( counter = regprescaler) then
      counter <= (others => '0');
      tc <= not tc;
    else
      counter <= counter + 1;
    end if;
  end if;
end process;

U_TC: CLKINT port map(tc, buf_tc);
- Processo di gestione free running counter
process( buf_tc, RESN ) begin
  if( RESN='0' ) then
    freecounter <= (others => '0');
  elsif( buf_tc'event and buf_tc='1') then
    if( freecounter = "11111111") then
      freecounter <= (others => '0');
    else
      freecounter <= freecounter + 1;
    end if;
  end if;
end process;

```

Listato 3

```
Comparator: process ( CLEARN, Clock )
begin
  if ( CLEARN = '0' ) then -- if CLEARN goes to zero it clears the EQUAL line
    RunningEqualStop <= '0';
  elsif (Clock'event and Clock = '0') then
    if RunningCounter > StopValue then
      RunningEqualStop <= '1';
    else
      RunningEqualStop <= '0';
    end if;
  end if;
end process;
PWMprocess: process ( CLEARN, StartPWM, EnablePWM, RunningEqualStop, Clock )
begin
  if (CLEARN = '0' ) then
    PWMout <= '0';
  elsif (Clock'event and Clock = '1' ) then
    if (EnablePWM = '1' and RunningEqualStop = '0') then
      PWMout <= '1';
    elsif (RunningEqualStop = '1' or EnablePWM = '0' ) then
      PWMout <= '0';
    end if;
  end if;
end process;
```

al simulatore Modelsim. In questa sezione, usando il file VHD di simulazione **prova_spi_actel_tbench.vhd**, abbiamo simulato la programmazione della SPI, in modalità master e conseguentemente una trasmissione di caratteri. Lanciando il simulatore sarà possibile analizzare i parametri e le tempistiche della sezione SPI (**figura 6**). Per la simulazione va ricordato che è possibile avere più file di stimolo disponibili ed è possibile la scelta di quale utilizzare cliccando con il tasto destro del mouse nel corner in alto a sinistra della casella stimulus (**figura 0**) e selezionando la voce organize stimulus. Per arricchire ulteriormente la simulazione (per esempio provare la generazione dei PWM), sarà sufficiente lanciare il generatore di stimoli WaveFormer cliccando sulla casella omonima presente nel project flow. Una volta aperto troverete la sezione già costruita (di default carica per la simulazione della SPI il file **prova_spi_actel_tbench.btim**) e potrete proseguire disegnando ulteriori cicli. Ricordate sempre che una volta finita la creazione dei nuovi cicli, dovrete effettuare il salvataggio non solo della parte grafica disegnata (estensione *.btim), ma anche la generazione del

file di simulazione vero e proprio. Per tale creazione è sufficiente selezionare dal menù file la voce Save as il formato Top level VHDL test bench.

Dal punto di vista fisico (quindi nella sezione Place and Route di libero) nel progetto già compilato, che potrete scaricare, troverete l'assegnamento dei pin fatto in modo arbitrario.

Qualora desideriate cambiare la loro posizione, è sufficiente che nell'ambiente DESIGNER facciate un doppio click sull'icona **IO attribute Editor** e nella videata, che vi verrà proposta, troverete tutti i pin del progetto con la relativa numerazione, che sarà da voi sostituibile. Sempre in questa area potrete decidere se inserire resistenze di Pull up o Pull down, aumentare la capacità di corrente di uno specifico pin oppure selezionare un funzionamento a basso slew rate.

Per coloro che invece volessero usare un differente package, è sufficiente dal menù Tools selezionare la voce Device selection e procedere di conseguenza.

Conclusioni

Per verificare il numero di gate consumati per questa applicazione, è suffi-

ciente selezionare nel Sw di Place e Route **Designer** il menu Tools -> Reports ->Status. Dall'analisi di questo report possiamo verificare che il numero delle macrocelle utilizzate è di 677 celle base (o Tile) e per il package QN48 33 I/O utilizzati su 34 disponibili. Sempre nel report status possiamo verificare l'utilizzo della A3PN030 all'88% della sua massima dimensione.

La conclusione di questo esempio ci porta a poter verificare che con un componente, che ha un costo di un paio di dollari, possiamo implementare non soltanto delle semplici sezioni di glue logic, come siamo abituati a vedere in molte applicazioni in queste fasce di costo, ma vere e proprie periferiche mediamente complesse, sempre restando su componenti di dimensioni e costi contenuti.

Solo come termine di paragone rispetto alle molte applicazioni basate su CPLD è utile pensare che selezionando il taglio immediatamente successivo alla A3PN030 (quindi A3PN060), con aumento di prezzo dell'ordine del dollaro, potremo raddoppiare le periferiche integrate e portare i PWM a 16b bit. ■