ELEKTRONIK TIDNINGEN

EXPERTARTIKEL

Kristian Saether
produktmarknadschef
Atmel

Så maxar du din MCU:s prestanda

Direkt minnesaccess, acceleratorer och händelsesystem – utvecklare känner ofta inte till dessa funktioner hos moderna MCU:er, trots att bär på dramatiska potentiella prestandaförbättringar.
Kristian Saether lär dig några knep för hur de kan utnyttjas.

Redaktör
Jan Tångring
jan@etn.se
0734-17 13 09

EMBEDDED EXPERT

Så maxar du din MCU:s prestanda

Direkt minnesaccess, acceleratorer och händelsesystem – utvecklare känner ofta inte till dessa funktioner hos moderna MCU:er, trots de dramatiska prestandaförbättringar de kan ge.

Kristian Saether, Produkt Marketing Manager, Atmel

Kristian Saether tog sin ingenjörsexamen på NTNU-universitetet i Trondheim år 2003. Han anställdes på Atmel året efter och idag är

MCU-kretsar utvecklas ständigt till alltmer komplexa system som förväntas utföra en större mängd olika uppgifter till lägre kostnad och med lägre effektförbrukning jämfört med tidigare generationer. För att kunna arbeta med de höga datahastigheter och frekvenser som krävs för ett processa inbyggda system i realtid måste dessa kretar uppnå en högre beräkningseffektivitet.

Den traditionella ansatsen att öka klockfrekvensen ger allt mindre och mindre prestandaförbättringar relativt effektförbrukningen. Riktiga förbättringar av verkningsgraden måste därför baseras på arkitektoniska innovationer.

Av dagens styrkretsar krävs att de ska kunna hantera många helt olika typer av uppgifter samtidigt: styrning i realtid, avkodning av bredbandiga kommunikationsprotokoll, bearbetning av massvolymer av data från sensorer, med mera. Det kostar alltför många processorcykler att kontinuerligt polla gränssnitten för att se om nya data har anlänt, och en sådan metod har också ofta en alltför lång maximal svarstid för att tillförlitligt kunna betjäna I/O- och periferienheter.

För de flesta inbyggnadsapplikationer är utvecklarna beroende av avbrott (interrupt) för att klara realtidskraven vid styrning av periferienheter. Men ett avbrott kan bara avgöra när en realtidshändelse har inträffat. Utvecklarna måste fortfarande direkt engagera CPUn för att läsa av I/O- och periferienheterna innan data går förlorade.

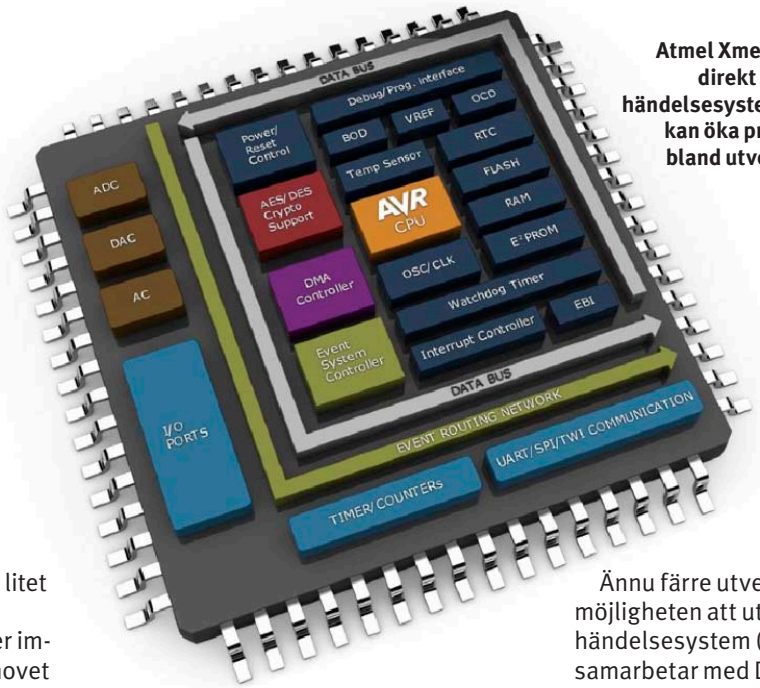
Att hantera ett avbrott avbryter potentiellt andra latenskänsliga uppgifter, lägger till kontextväxlingsoverhead och skapar olika esoteriska problem som att hantera latenser när flera avbrott uppträder samtidigt. Allt detta minskar förutsägbarheten och processorns verkningsgrad.

För att kunna hantera de höga datahastigheter och frekvenser som förekommer i realtids I/O- och periferienheter måste MCU-kretsarna uppnå högre beräkningseffektivitet. Detta kan dock inte uppnås genom ökad klockfrekvens – eftersom det skulle medföra högre effektförbrukning – utan måste ske genom ändringar av MCU-arkitekturen.

Mer specifikt har MCU-kretsarna börjat få integrerade hjälpprocessorer som avlastar specifika uppgiftsblock, flerkanals DMA-styrkretsar som förenklar icke resurskrävande minnesaccess samt integrerade händelsesystem som routar signaler mellan interna subsystem för att avlasta I/O- och avbrottshanteringen.

Integrerade hjälpprocessorer har blivit ganska vanliga i olika typer av inbyggnads-MCU:er. Acceleratorer för kryptering och TCP/IP är två vanliga typer av hjälpprocessorer. De avlastar antingen kompletta uppgifter, eller så assisterar de vid mer datorintensiva delar av komplexa algoritmer.

En krypteringsmotor minskar exempelvis antalet beräkningssteg som krävs i CPU:n för en AES-operation från tusentals till hundratals, och en TCP/IP-motor gör det möjligt att terminera



Atmel Xmega har kryptoaccelerator, direkt minnesaccess (DMA) och händelsesystem. Det är funktioner som kan öka prestanda dramatiskt, men bland utvecklare är de ofta okända.

en Ethernetanslutning med mycket litet CPU-overhead.

Dessutom förenklar acceleratorer implementeringen och eliminerar behovet av att skriva omfattande drivrutiner, och ger istället utvecklarna tillgång till denna avancerade funktionalitet med hjälp av enkla API:er.

Teknologier för DMA- och händelsehantering är mindre kända, och därför används de mer sällan av utvecklare. DMA-styrkretsar avlastar hanteringen av datakopiering från CPU:n genom att lägga dataaccesser – till exempel från periferiregister till internt eller externt SRAM – i bakgrunden.

En utvecklare kan till exempel konfigurera en DMA-styrkrets att förladda ett datablock i RAM på chipet så att det finns tillgängligt för snabb access innan CPU:n behöver det. På så sätt elimineras väntetider och fördröjningar som beror på databeroenden. Alternativt kan en DMA-styrkrets ta över merparten av bördan av att hantera periferienheter för kommunikation (se tabell 1).

Besparingarna i klockcykler när man använder en DMA-styrkrets kan vara betydande: Det har förekommit fall där inbyggnadsutvecklare upptäckt att de inte får plats med en applikation inom resursgränserna för MCU:n. Men när tillverkaren upplyst om möjligheten att använda DMA, har utvecklaren plötsligt fått klockcykler över, upp till 30 till 50 procent över hela systemet. För många utvecklare är det först när de ställs inför en oöverstiglig beräkningsmur som de för första gången upptäcker vilken outnyttjad potential de haft tillgång till redan från början.

Ännu färre utvecklare känner till möjligheten att utnyttja så kallade händelsesystem (event systems). Sådana samarbetar med DMA-styrkretsar för att ytterligare avlasta CPU-cykler och minska den totala effektförbrukningen. Ett händelsesystem är en buss som kopplar interna signaler från en periferienhet på en MCU till en annan – när en händelse inträffar i en periferienhet triggar det att en aktivitet skall utföras i en annan periferienhet – utan att CPU:n är inblandad. Och detta med en fördöjning på bara två klockcykler. Systemet liknar människokroppens reflexer som kan utlösa aktiviteter utan att först konsultera hjärnan, till exempel att dra handen ur elden.

Närmare bestämt fungerar det såhär: händelsesystemet routar signaler genom MCU:n med hjälp av ett speciellt nätverk som förbinder CPU:n, databussen, periferienheterna och DMA-styrkretsen. I normala fall måste periferienheterna

UART Transfer Speed	CPU usage with	CPU usage without
9.6	0.01	0.26
19.2	0.01	0.52
38.4	0.03	1.04
57.6	0.04	1.57
115.2	0.08	3.14
1200	0.85	34.15
3500	5.17	99.59

Tabell 1
DMA-styrkretsar kan ta över bördan att hantera periferienheter för kommunikation.

avbryta CPU:n för att initiera aktiviteter, som exempelvis avläsning.

Genom att routa händelserna direkt mellan periferienheterna avlastar händelsesystemet dessa avbrott från CPU:n. Utvecklarna kan konfigurera periferienheterna att följa olika händelsekanaler, och kan på så sätt definiera vilken händelseroutning som krävs för att uppfylla applikationens specifika behov.

Kombinationen av direkt minnesaccess som samarbetar med händelsesystem ger utvecklarna möjlighet att avlasta kompletta uppgifter, ungefär som med en hjälpprocessor. Men en viktig skillnad är att hjälpprocessorer inte är programmerbara. De implementerar bara väl definierade uppgifter i hårdvara och kan som hägst konfigureras.

Möjligheten att programmera DMA-styrkretsar och händelsesystem gör att de passar till en mängd olika uppgifter, från enkla till komplexa.

I fallet där DMA används med ett händelsesystem hanterar DMA överföringen av data genom MCU:ns arkitektur, medan händelsesystemet styr när dessa överföringar sker, med låg latens och med en hög noggrannhet. Man kan uttrycka det så att händelsesystemet ser till att de värden som hanteras av DMA samplas eller matas ut vid rätt tidpunkt och frekvens.

Fig 1 visar ett blockdiagram över hur händelsesystem och DMA samarbetar. AD-omvandlaren (ADC) samplar en sensor. En intern räknare har anpassats till samplingsfrekvensen och tillhandahåller regelbundna och exakta klockintervall.

Istället för att avbryta CPU:n för att läsa av ADC:n initierar händelsesystemet direkt en avläsning av denna. Detta gör att samplingsfrekvensen blir ytterst noggrann relativt MCU:ns klocka. När ADC:n avslutat omvandlingen triggar den DMA att lagra värdet via händelsesystemet.

Händelsehanteringen kan utökas till att innehålla fler händelser och ansluta fler periferienheter så att mer komplexa konfigurationer skapas. Som exempel kan en insignal (händelse 1) trigga en ADC att sampla (händelse 2) och sända värden till DMA (händelse 3) tills DMA-bufferten är full (händelse 4). I denna konfiguration får CPU:n en avbrottssignal först när det finns en buffert som är full av data som skall bearbetas.

Såväl DMA-styrkretsar som händelsesystem stöder också flera kanaler. Det ger utvecklaren möjlighet att konfigurera en sammankopplingsenhet som arbetar parallellt med huvud-CPUn. Det gör att flera samtidiga realtidsuppgifter kan koordineras på ett deterministiskt vis.

Determinism spelar en viktig roll när det gäller att begränsa latenser och hantera reaktionstiderna i realtidssystem. Ju mer deterministiskt systemet är, desto mer konsekvent kommer dess reaktionstider att vara. En viktig faktor som påverkar determinismen är hur många avbrott systemet måste hantera samtidigt. Generellt gäller att när antalet avbrott ökar så försämras determinismen.

Låt oss se på ett system som har ett enda avbrott som avslutas inom 50 cykler. Latensen för ett sådant avbrott är genomgående omkring 50 klockcykler. Observera att även de enklaste avbrott använder omkring 50 cykler på att låta MCU:n spara kontext för ett begränsat antal register, anropa en periferienhet, spara data, återställa kontext och tömma rörledningen.

Men det är inte haneringen av ett ensamt avbrott som skapar de största problemen när det handlar om determinism och latens. De största problemen att uppfylla realtidskraven uppstår istället när många avbrott inträffar samtidigt.

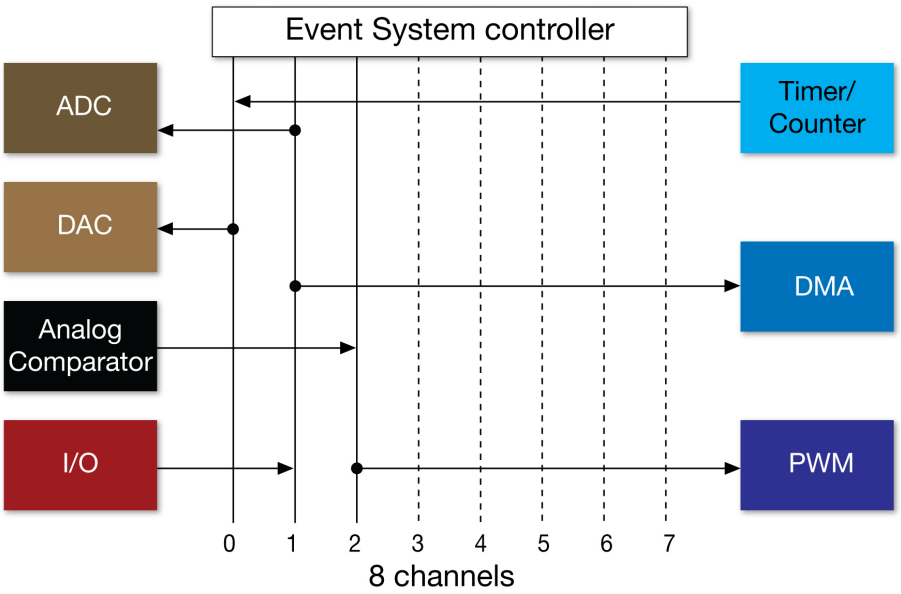
Antag till exempel att systemet tar emot en högprioriterad avbrottssignal som kräver 75 cykler. Latensen för det första avbrottet påverkas då, eftersom uppgiften med högre prioritet går före. Uppgiften med lägre prioritet får nu en latens som ligger i området 50 till 125 cykler.

I takt med att allt fler avbrott signaleras, kommer latensen för avbrott med lägre prioritet att öka när determinismen minskar. En 50-cyklers uppgift kan avbrytas många gånger och kräva hundratals eller tusentals cykler innan den avslutas. Denna faktor är viktig att komma ihåg, eftersom alla avbrott förstås inte kan ha högre prioritet än alla andra.

Determinismen påverkar direkt reaktionstiderna, tillförlitligheten och noggrannheten. Om man vet att latensen har ett fixt värde mellan 50 eller 500 cykler kan man ta hänsyn till detta under bearbetningen. Men om latensen fritt kan variera mellan 50 och 500 cykler är det bästa man kan göra att förutsätta att den typiskt är till exempel 200 cykler, och betrakta varje avvikelse som ett fel. I värsta fall kan latensen närma sig de specificerade realtidsgränserna för systemet, och därmed hota tillförlitligheten.

Att minska det potentiella antalet samtidigt uppträdande avbrott – inklusive sällsynta sådana – med hjälp av en DMA-styrkrets och ett händelsesystem, kan väsentligt öka systemets determinism och dessutom minska dess latens.

Högre determinism ger också andra viktiga fördelar, som högre noggrann-



Figur 1
Här samarbetar en DMA-styrkrets och ett händelsesystem om att avlasta CPU:n från att hantera periferienheter.
En intern räknare bestämmer samplingsfrekvensen. Den levererar regelbundna och noggranna intervaller alternativt en insignal (händelse 1), som kan trigga en ADC att sampla (händelse 2) och sända värdet till DMA (händelse 3) tills DMA-bufferten är full (händelse 4).
I denna konfiguration får CPU:n en avbrottssignal först när det finns en buffert som är full av data som skall bearbetas.

het. För att förstå hur determinismen kan påverka noggrannheten kan vi tänka oss ett kraftövervakningssystem som maximerar AC-verkningsgrad vid drift av svåra laster som motorer.

Den högsta energin finns tillgänglig när spänningen har ett toppvärde och är i fas med strömmen, och därför är det just då som systemet bör dra mest ström. Omvänt, ju närmare spänningen kommer noll (det vill säga nollgenomgången), desto mindre effekt finns tillgänglig och desto mindre bör strömförbrukningen vara.

Genom att använda PFC (Power Factor Correction) kan man förbättra verkningsgraden. Stora kondensatorer kopplas då in och ur för att justera belastningen så att strömmen och spänningen hålls i fas. Vanligen används en komparator för att detektera nollgenomgången. När spänningen sjunker under eller stiger över ett inställt tröskelvärde växlar komparatorn tillstånd.

Men man behöver inte låta komparatorn trigga ett avbrott och tvinga CPU:n att switcha kondensatorerna. Istället kan man låta händelsesystemet routa komparatorhändelsen direkt till den Timer/Counter-utgång som styr switchningen, utan att CPU:n behöver medverka.

Avbrottslatensen för lågprioriterade uppgifter som PFC kan uppgå till tusentals cykler, beroende på hur många avbrott med högre prioritet som uppstår samtidigt. Högre latens medför att kondensatorerna switchas senare än vid den optimala tidpunkten, vilket minskar den totala verkningsgraden avsevärt. Latensen vid händelseroutning uppgår som jämförelse till högst två cykler.

Låt oss ställa dessa tal i relation till MCU:ns klockfrekvens. Om MCU:n klockas med 32 MHz ger en latens på två cykler ett försumbart fel (2/32M). Om å andra sidan latensen uppgår till tusentals cykler kan den i väsentlig grad påverka noggrannheten hos högfrekventa uppgifter som själva måste bearbetas med några tusen klockcyklers tidsmellanrum.

Observera att denna latens kan minskas till storleksordningen 50 cykler genom att man gör avbrottet till en högprioriterad uppgift. Men detta leder till att man tilldelar prioritet baserat på noggrannhetskrav, snarare än på hur viktig en funktion är för systemet. Dessutom flyttas onoggrannheterna på grund av brist på determinism över till andra uppgifter.

Högre noggrannhet har också betydelse vid generering av signaler, och inte bara vid sampling av dem. Säg att en vågform på 100 kHz skall skapas. Om man använder avbrott kommer noggrannheten hos vågformen att påverkas av den variabla latensen i förhållande till signalfrekvensen – något långsammare eller snabbare beroende på kontextväxling och de övriga avbrott som samlats på hög.

Observera att vågformen visserligen kommer att vara noggrann i genomsnitt. Men vad som i många fall har betydelse är den relativa skillnaden mellan två på varandra följande samplingar.

Att generera signaler har blivit en allt vanligare uppgift i inbyggnadsapplikationer. Signaler används för att generera ljud, hantera spänningsomvandlare och styra aktuatorer i industriella applikationer, liksom för en myriad andra funktioner. Ju högre frekvens signalen har, desto högre blir belastningen på CPU:n när avbrott används, och desto större blir risken för ökad latens för andra uppgifter.

För händelser som inträffar med högre frekvens blir CPU-belastningen en viktig faktor. Snabba sensorer måste till exempel vara klara med insamlandet av ett sampel när nästa sampel ligger klart, detta för att inte data skall förloras. Det kan gälla till exempel på en flödesmätare, ett fleraxligt positioneringssystem eller ett instrumenteringssystem som samlar in 2 MSa/s för att kunna göra snabba och noggranna mätningar. Ett sådant system kommer att förbruka tiotals till hundratals miljoner klockcykler varje sekund, bara för att samla in samplen. Med ett händelsesystem och en DMA-styrkrets kommer alla dessa cykler att avlastas från CPU:n, så att denna istället kan bearbeta samplen och inte bara buffertlagra dem.

Även om vi antar att det handlar om en enkel uppgift som bara kräver 50 klockcykler, med overhead för kontextväxling, så medför detta att man avlastar 100 Mcykler från CPUn. Därför använder många system en separat MCU för att

Här är fler exempel på hur händelsesystem och DMA kan lösa problem kopplade till krav på snabb bearbetning.

- Noggrann tidsstämpling: Genom att tidsstämpla sampel förenklar man synkroniseringen av signaler till externa händelser. Med två cyklers latens blir tidsstämplarna mycket noggrannare än om avbrotten haft en latens i storleksordningen tusentals cykler.

- Översampling: Ett sätt att öka en sensors upplösning är att översampla. Om man till exempel delar ned en räknare med 16 så samlar man in 16 gånger fler sampels, vilket ökar den totala noggrannheten. Eftersom CPU:n inte är direkt inblandad i att samla in och lagra samplen går det att använda översampling utan större uppoffringar.

- Dynamisk frekvens: Vissa applikationer kräver högre avkänningsnoggrannhet bara vid speciella tidpunkter eller under speciella arbetsförhållanden. En vattenmätare kan till exempel sampla snabbare när flödeshastigheten ändras snabbt, och sedan gå ned igen när flödet stängs av eller är konstant. Samplingsfrekvensen kan enkelt ändras utan att detta påverkar reaktionstiderna.

- Minskad stackstorlek: En extra effekt av att minska det potentiella antalet samtidiga avbrott är att det går att använda en mindre stack. Vid varje avbrott måste kontexten sparas genom att kanske dussintals register flyttas till stacken. Om man då tar bort flera kontextlager kan man väsentligt minska den nödvändiga storleken på stacken. Detta kan leda till att man behöver en mindre mängd RAM i sin applikation.

- Immunitet mot skalning: Eftersom olika MCU:er stöder olika antal periferienheter kan antalet avbrott inom en applikation variera med vilken prisnivå MCU:n ligger på. Inom samma MCU-familj stödjer ett mer avancerat system inte bara fler funktioner utan också fler avbrott, vilket sänker determinismen. Att migrera en konstruktion till en mer integrerad MCU kan därför påverka signallatensen och därmed noggrannheten, både för sampling och utmatning.

- Enkla mjukvaruändringar: Om man har händelsehantering som gör att CPU:n inte behöver intervenera, kan ändringar i mjukvaran utföras utan att detta påverkar realtidsegenskaperna. Även om CPU:n behöver mer tid för att hantera tillkommande funktioner kommer händelsehanteringen och svarstiderna att förbli exakt samma. Utan denna lösning kan det vara svårt att implementera ändringar under en produkts livslängd när det gäller realtidsapplikationer.

		Active		Idle Mode		Power save		
Atmel XMEGA	Cycles	Time	Current (mA)	Time	Current (mA)	Time	Current (mA)	Average current
No DMA, No Event	1200000	10%	3.80	0%	1.90	90%	0.55	380.495
DMA on	960000	6%	3.80	4%	1.90	90%	0.55	304.495
DMA and Event	840000	5%	3.80	5%	1.90	90%	0.55	285.495

Tabell 2
Ett händelsesystem och en DMA-styrkrets ökar inte bara CPU:ns kapacitet och prestanda, utan kan också väsentligt minska effektförbrukningen – beroende på applikation – genom att ge MCU:n möjlighet att gå ned i tomgångs- eller sömnläge oftare. Med tanke på att strömförbrukningen i aktivt läge är avsevärt högre än i tomgångsläge kan redan några få procents förkortning av tiden i aktivt läge ge avsevärda effektbesparingar.

hantera individuella högfrekventa sensorer och motorer.

Det finns en myriad knep som en inbyggnads-MCU kan ta till för att till exempel minska effektförbrukningen, öka noggrannheten eller förbättra användargränssnittet. Många av dessa knep är enkla övervakningsuppgifter som bara går ut på att kontrollera ett visst värde. En batterimonitor övervakar och noterar att spänningen sjunkit under en viss nivå, och triggar en avstängning för att spara undan applikationsdata medan det ännu finns ström kvar för att klara detta.

Att förbättra användarens upplevelse är en viktig differentierande faktor i många konsumentprodukter. Ett händelsesystem kan till exempel ge bättre responsivitet på en tangentnedtryckning eller en signal från en periferienhet och ge en reaktion inom 2 cykler.

Jämför detta med den responsivitet man uppnår om man använder avbrott. Avbrott kräver dessutom att systemet efteråt återgår till ett aktivt läge, vilket sänker beräkningseffektiviteten. Detta är en orsak till att utvecklare ofta ökar timerintervallen, med försämrad responsivitet som straff.

Användandet av interrupt har medfört att kostnaden för att implementera knep av detta slag har varit för hög, räknat i CPU-operationer, förlängd latens och minskad determinism. Med hjälp av ett händelsesystem och en DMA-styrkrets kan man implementera sådana funktioner, samtidigt som man effektivt undviker att belasta MCU:n. Detta inte bara minskar antalet avbrott som systemet måste hantera, utan förenklar också implementeringen och hanteringen av uppgiften.

Som exempel ska vi titta på en applikation som spelar upp ett varningsmeddelande för användaren under ett speciellt villkor. Den färdiga ljudfilen kan lagras i en buffert och matas till högtalaren via en lämplig periferienhet med hjälp av DMA.

Händelsesystemet, som använder en timer, garanterar att datahastigheten här blir exakt 44,056 kHz. Eftersom denna frekvens är noggrann och konstant får man som en sidoeffekt att även ljudkvaliteten ökar. Ur prestandasynvinkel gäller att så snart som DMA och händelsesystem har konfigurerats så kommer CPU:n att fullständigt vara avlastad denna uppgift.

Att påstå att knep av detta slag blir ”gratis” är en överdrift. Men att implementera dem på detta vis gör att de blir användbara i många fler olika slags applikationer. Kombinationen av hjälpprocessorer, en DMA-styrkrets och ett händelsesystem ger MCU:n frihet att enbart sköta signalbehandling, istället för att förbruka merparten av sina resurser på klockcykelintensiv signalinsamling.

Resultatet är att CPUn sparar in bearbetningskapacitet som kan användas för signalbehandling. Därigenom blir det möjligt att låta en och samma MCU hantera flera stycken högfrekventa uppgifter, istället för bara en enda. Det förenklar också systemkonstruktionen och gör att fler uppgifter kan implementeras till lägre kostnad på en och samma MCU. Det blir enklare att korrelera många olika signaler, och man får högre effektverkningsgrad.

I många applikationer kan möjligheten att stödja flera uppgifter leda till en viktig produktdifferentiering. Så kan till exempel en applikation för motorstyrning som använder en DMA-styrkrets och ett händelsesystem friställa tillräckligt mycket resurser hos MCU: för att implementera avancerade funktioner som PFC utan att öka systemets komponentantal.

Händelsesystemet kan inte bara ge MCU:n högre prestanda genom att avlasta den från avbrott, utan dessutom minska effektförbrukningen. Med upp till en faktor sju, beroende på applikationen. Tabell 2 visar effektvärden för en applikation som kräver 1,2 Mcykler/s, med Atmels MCU XMEGA som exempelkrets.

Vid 12 MHz är MCU:n i aktivt läge under 10 procent av tiden och i standbyläge resten av tiden. Genom att implementera en DMA-styrkrets och ett händelsesystem minskar man det antal cykler CPU:n måste exekvera varje sekund, vilket gör att MCU:n kan gå ned i tomgångsläge. Med tanke på att strömförbrukningen i aktivt läge är avsevärt högre än i tomgångsläge kommer bara några få procents förkortning av tiden i aktivt läge att ge avsevärda effektbesparingar. ■

Slutsatser

Inbyggnads-MCU:er får ständigt allt högre prestanda genom arkitektoniska förändringar som förbättrar den totala CPU-kapaciteten. Här är tre exempel: hjälpprocessorer som avlastar väldefinierade beräkningsintensiva uppgifter från CPU:n, DMA-styrkretsar som befriar CPU:n från att flytta kring data i systemet, och händelsesystem som eliminerar de flaskhalsar som förknippas med avbrott som triggas med hög frekvens. Genom att minska antalet samtidiga avbrott kan utvecklaren öka systemets determinism och därmed få lägre latens, ökad upplösning och högre noggrannhet i signalerna, högre konsistens och förutsägbarhet samt högre systemtillförlitlighet. Resultatet är att en och samma MCU kan utföra samma arbete som flera äldre MCU:er kunde och detta till lägre systemkostnad och med minskad effektförbrukning.