Räcker 8 bitar eller behöver jag 32?

EXPERTARTIKEL

Här – en guide som hjälper dig i valet

Av Josh Norem, Silicon Labs

Josh Norem är systemingenjör och arbetar med styrkretsar och trådlösa produkter på Silicon Labs. Han började på Silicon Labs år 2006, först som testingenjör, därefter har han arbetat i flera andra teknikintensiva roller. Innan Silicon Labs arbetade Josh på AMD med x86-arkitekturen. Dessförinnan var han på Texas Instruments där han arbetade med DSP:er.

Varför skulle jag vilja ha en 8-bitars MCU? Den frågan får jag då och då, och här ska jag försöka ge ett svar. För en sak är säker. Du som utvecklar får ofta ett enklare jobb, och slutprodukten bli ofta bättre om du bara lägger några timmar i början av utvecklingsarbetet på att utvärdera vilken arkitektur som passar just din tillämpning bäst.

Den här artikeln är tänkt som en guide till hur du väljer mellan 8- och 32-bitars MCU-arkitekturer. De flesta exemplen kring 32-bitare handlar om ARM Cortex M-kretsar, vilka beter sig väldigt lika från olika MCU-leverantörer. Bland 8-bitare finns det många fler arkitekturvarianter, men för jämförelsen använder jag den vitt spridda och välkända 8-bitarsarkitekturen 8051, som fortfarande är populär bland embeddedutvecklare.

Alla som har ett enkelt svar på frågan om ARM eller 8051 är bäst för en tillämpning försöker sälja något. Frågan ”Vad är bäst, Cortex M eller 8051?” är inte logisk. Det är som att fråga ”Vad är bäst, en gitarr eller ett piano”. En mycket bättre fråga är ”Vilken MCU hjälper mig bäst att lösa problemet jag jobbar på?”

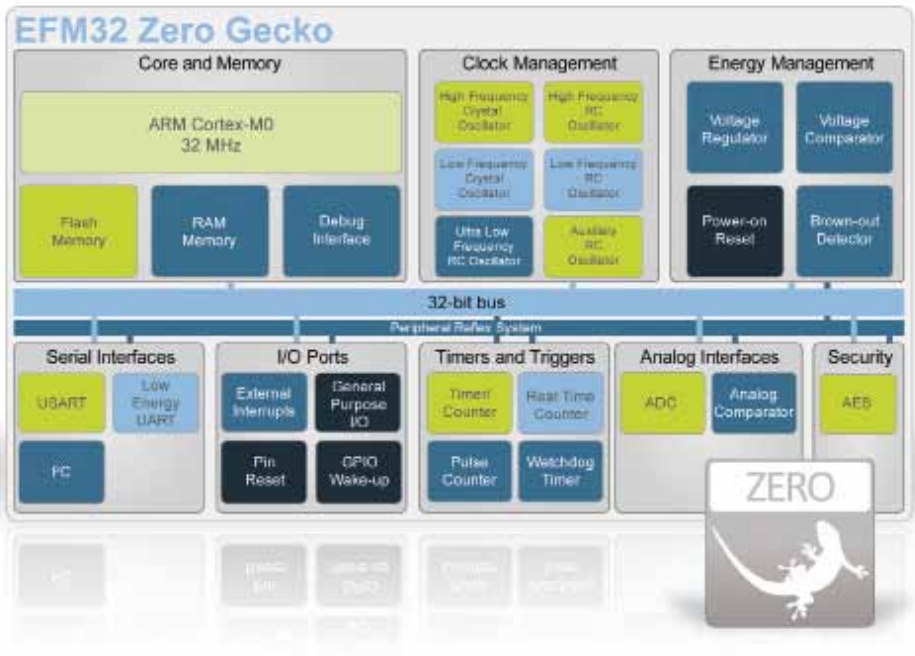
FÖR ATT JÄMFÖRA måste man mäta. Det finns en hel del verktyg att välja mellan. Jag har försökt välja scenarier som jag tror ger den mest rättvisa jämförelsen och som är mest representativa för erfarna utvecklare. Siffrorna för ARM nedan har genererats med kompilatorn GCC och biblioteket nano C med optimeringsflagga -03.

Jag har inte försökt optimera kod. Jag har helt enkelt implementerat den mest uppenbara normal-kod som 90 procent av alla utvecklare skulle använda eftersom jag är mer intresserad av vad den genomsnittliga utvecklare kommer att, nå än optimala förhållanden.

Innan vi börjar jämföra arkitekturer är det viktigt att förstå att alla MCU:er inte är likadana. Om vi ställer en modern ARM Cortex Mo+ mot en 8051 som är 30 år gammal, så har 8051:an inte en chans vid en prestandajämförelse. Dessbättre har ett antal leverantörer successivt investerat i 8-bitarsprocessorer. Silicon Labs EFM8-portfölj av 8051-baserade MCU:er är exempelvis långt mer effektiv än den ursprungliga arkitekturen. Likaså har utvecklingsprocessen moderniserats. Resultatet är en 8-bitars kärna som i många tillämpningar kan hävda sig mot en Mo+ eller M3-kärna, och ibland till och med prestera bättre.

Även utvecklingsverktygen är viktiga. För att utveckla firmware krävs en fullfjädrad utvecklingsmiljö (IDE), ett användbart firmware-bibliotek, omfattande exempel, heltäckande utvärderings- och startpaket, liksom hjälpprogram som exempelvis förenklar hårdvarukonfiguration, biblioteckshantering och programmering för produktion.

GENERELLT KAN MAN SÄGA att Cortex M-kärnor är de bäst lämpade för stora system (>64 kB kod), medan 8051-enheter vinner i mindre (<8 kB kod). Däremellan kan båda fungera bäst, allt beroende på vad systemet gör.



Ofta spelar periferienheterna en viktig roll. Behöver du tre UART:ar, en LCD-styrenhet, fyra timrar och två ADC:er, hittar du knappaset en passande 8-bitare. I det område där båda arkitekturerna är möjliga alternativ, handlar valet om en avvägning mellan den utvecklarvänlighet som en ARM-kärna ger och det lägre priset och den mindre storleken som fås med en 8051-enhet.

Cortex M-arkitekturens enhetliga minnesmodell i kombination med fullt stöd för C99 i alla vanliga kompilatorer, gör det mycket enkelt att skriva firmware. Till detta hör en enorm uppsättning bibliotek och kod från tredje part. Enkelhetens baksida är förstås kostnaden. Samtidigt är enkelhet viktigt om tillämpningen har hög komplexitet, tiden till marknad är knapp eller firmwareutvecklaren är oerfaren.

DET ÄR LÄTT ATT HITTA 8-bitare med 2 kB flash och 512 byte RAM, medan 32-bitare sällan har minnen mindre än 8 kB respektive 2 kB. Denna spännvidd i minnesstorlek ger systemutvecklare möjlighet att hitta billiga lösningar i system som inte behöver så mycket resurser. Av samma anledning gynnas tillämpningar som är kostnadskänsliga eller bara behöver litet minnesutrymme, av en 8051-lösning.

8-bitare har i allmänhet också en fördel när det gäller fysisk storlek. Ta till exempel den minsta 32-bitars MCU:n från Silicon Labs som kommer kapslad i en QFN. Den mäter 4 mm × 4 mm, medan 8051-baserade MCU:er är så små som 2 mm × 2 mm i QFN-kapsling. Storleksskillnaden är mindre i CSP-format, men då höjs priset och monteringen blir svårare. I tillämpningar där utrymmet är väldigt begränsat är en 8051 ofta det enda alternativet.

En 8051 använder vanligtvis flash och RAM mer effektivt än en Cortex M, vilket är en av de viktigaste anledningarna till det lägre priset. Fast ju större systemet är, desto mindre kommer detta att inverka. Samtidigt utnyttjar 8-bitarsarkitekturer inte alltid minnet mest effektivt. I vissa situationer är en ARM-kärna lika effektiv eller effektivare. Ta exempelvis 32-bitars matematiska operationer; de kräver endast en instruktion hos en ARM, men flera instruktioner hos en 8051.

ARM-arkitekturen har två stora nackdelar när flash- och RAM-minnet är litet: lägre kod/yteffektivitet och sämre förmåga att förutsäga hur RAM:et används.

Vad gäller kod/yteffektiviteten är 8051-instruktioner är på 1, 2 eller 3 byte, medan ARM:s är på 2 eller 4. I genomsnitt har alltså en 8051:a mindre instruktioner. Å andra sidan kan en ARM-kärna göra mer med en instruktion. Exemplet ovan med 32-bitars aritmetik är bara ett av många exempel. I praktiken innebär den mindre instruktionsbredden endast måttligt högre koddensitet för 8051-systemet.

I SYSTEM SOM ANVÄNDER distribuerad åtkomst av variabler är ARM-arkitekturens load/store-konstruktion ofta viktigare än instruktionsbredden. Betrakta exempelvis implementationen av en semafor, där samma variabel minskas (allokeras) och ökas (frigörs) på olika platser i koden. En ARM-kärna måste ladda variabeln i ett register och bearbeta den för att sedan lagra den igen, vilket tar tre instruktioner. En 8051-kärna kan arbeta direkt i minnet och kräver bara en instruktion.

I situationer då en variabel ska bearbetas i större omfattning blir ARM-kärnans extra instruktioner för att ladda och lagra försumbara, medan 8051 är klart effektivare när bara lite arbete utförs per gång.

Även om semaforer är ovanliga i embeddedprogramvara, så används enkla räknare och flaggor i ofta i styrtillämpningar och dessa beter sig på samma sätt. En uppsjö MCU-kod hör till denna kategori.

Nästa pusselbit handlar om att ARM-processorn utnyttjar stacken mycket flitigare än 8051. Typiskt lagrar en 8051 bara en returadress (2 byte) på stacken vid funktionsanrop, och statiska variabler får ersätta data på stacken. Sådana funktioner blir visserligen inte reentrant, men å andra sidan blir stackutrymmet litet och ganska förutsägbart, vilket har betydelse i MCU:er med begränsat RAM.

SOM ETT ENKELT EXEMPEL skrev jag en funktion som tar en 32-bitars parameter och i sin tur anropar en funktion med två 16-bitarsparametrar. När jag sedan mätte stackdjupet i den senare funktionen visade det sig att M0+-kärnan använde 48 byte medan 8051-kärnan enbart nyttjade 16 byte plus 8 byte statiskt RAM, vilket totalt gav 24 byte. I större system blir denna skillnad försumbar, men i ett system som enbart har 256 byte RAM blir den viktig.

Vi har nu målat upp grundförutsättningarna. Det är dags för en mer detaljerad analys av tillämpningar där de olika arkitekturerna utmärker sig och där de allmänna riktlinjerna inte håller.

Det finns en märkbar skillnad i fördröjning vid avbrotts- och funktionsanrop mellan de två arkitekturerna, där en 8051 är snabbare än en Cortex M. Samtidigt kan periferienheter på APB-bussen (Advanced ▶

Peripheral Bus) påverka fördröjningen eftersom data måste flyta mellan APB- och AHB-bussen (AMBA High-Performance Bus). Många Cortex M-baserade MCU:er kräver dessutom att APB-klockan delas ner när klockfrekvensen är hög, vilket ökar fördröjningen till periferienheterna.

Jag skapade ett enkelt experiment där ett avbrott utlöstes av en IO-anslutning. Avbrottet sköter en del signalering och uppdaterar en flagga baserat på vilken anslutning som utför avbrottet. Jag mätte parametrarna enligt nedan.

Parameter	ARM	8051
ISR Entry latency (T1-To)	1,09	0,94 µs
Min pulse width (T2-T1)	0,09	0,08 µs
ISR Execution Time (T3-T1)	1,09	0,74 µs
ISR Exit Time (T4-T3)	0,83	0,57 µs
TOTAL	**3,10**	**2,53 µs**

Avbrottsrutinen ISR (Interrupt Service Routine) har kortast start- och sluttid hos 8051-kärnan. När ISR blir större och tiden för exekvering ökar, blir dessa fördröjningar däremot obetydliga. Så ju större systemet blir, desto mindre fördel har 8051. ARM-kärnan gör dessutom ett snabbare jobb om ISR innebär att en stor mängd data ska skyfflas, eller om det utförs heltalsaritmetik på tal bredare än bitar.

EN 8051-KÄRNAS STYRKA är styrkod där åtkomsten av variabler är spridd och en mängd styrlogik används. 8051:an bearbetar också 8-bitars data mycket effektivt, medan en Cortex M utmärker sig vid databehandling och aritmetik på 32 bitar. Till detta hör att en ARM kan flytta 4 byte i taget, medan 8051 endast flyttar 1 byte i taget. Det betyder att en ARM-baserad lösning är mest lämpad i tillämpningar som i första hand strömmar data från en plats till en annan, exempelvis från UART till USB.

Betrakta detta enkla experiment. Jag kompilerade funktionen nedan på båda arkitekturerna med variabelstorlekarna 8, 16 och 32 bitar.

```
uint32_t funcB(uint32_t testA, uint32_t testB){
        return (testA * testB)/(testA - testB)
}
```

Data type	32 bit (-03)	8 bit
uint8_t	20 bytes	13 bytes
uint16_t	20 bytes	20 bytes
uint32_t	16 bytes	52 bytes

I takt med att datastorleken ökar, kräver 8051:an mer och mer kod för att göra jobbet. Vid 16 bitar är det dött lopp, men 32-bitarskärnan gynnas en aning av den högre exekveringstakten då den i allmänhet kräver färre klockcykler vid samma kodstorlek. Denna jämförelse är dock bara giltig när man kompilerar ARM-koden med optimering – annars blir koden flera gånger större.

Genom detta kan man inte dra slutsatsen att 8051:an är olämplig för alla tillämpningar som skyfflar mycket data eller arbetar med 32-bitars aritmetik. I många fall kommer ARM-kärnans effektivitet att uppvägas av annat. Ta exempelvis en UART till SPI-brygga. Den tillbringar största delen av tiden med att kopiera data mellan periferienheter, en uppgift som en ARM-kärna gör mycket effektivare. Samtidigt är det en mycket liten tillämpning, sannolikt tillräckligt liten för att rymmas i 2 kB.

ÄVEN OM EN 8051 ÄR mindre effektiv har den processorkraft nog för att hantera höga datatakter. I ovanstående exempel kommer ARM-kärnans extra processorcykler troligen inte att utnyttjas fullt ut, då är det mer värdefullt att spara pengar på ett mindre flash och ett mindre fotavtryck. Om ARM-kärnan däremot kan nyttja sina extra cykler på något användbart, så svänger fördelen igen. Med detta vill jag säga att det är viktigt att beakta varje arkitekturs styrka inom ramen för vad systemet som utvecklas ska hantera.

8051-kärnan har olika instruktioner för att nå kod (flash), IDATA (internal RAM) och XDATA (externt RAM), till skillnad mot ARM som har enhetlig minnesmappning. För att erbjuda effektiv kodgenerering kommer en pekare i 8051-koden att deklarera vilken del i minnet den pekar på. Ibland använder man en generisk pekare som kan peka på alla utrymmen, denna typ av pekare är ineffektivt att komma åt.

Segmentsspecifika pekare fungerar i de flesta fall, medan generiska pekare kan komma väl till pass när du skriver kod utan att känna till exakt var den kommer att användas. Om detta är vanligt i en tillämpning börjar 8051:an att förlora sin fördel i effektivitet.

OFTA HAR JAG KONSTATERAT att förekomsten av aritmetik talar för ARM och förekomsten av styruppgifter talar för 8051. Fast ingen tillämpning fokuserar ju enbart på matematik eller styrning, så hur kan vi karaktärisera en tillämpning i stora drag och räkna ut vilket val som är mest lämpligt?

Ta en hypotetisk tillämpning som består av 10 procent 32-bitars aritmetik, 25 procent styrkod och 65 procent allmän kod (general purpose) som inte tydligt faller in i 8- eller 32- bitskategorin. Dessutom antar vi att ett litet kodavtryck och pris är viktigare än exekveringshastighet. Detta ger nätt om jämnt ger en fördel för 8051-kärnan när det gäller allmän kod och en viss tydlig fördel för styrkoden. ARM-kärnan har å andra sidan övertaget när det gäller 32-bitars aritmetiken, men eftersom den delen är inte så stor här, så är resultatet är att 8051-kärnan passar bäst.

Om vi ändrar förutsättningarna till att exekveringshastighet är viktigare än kostnad, så drar ingen av arkitekturerna fördel av den allmänna koden. ARM-kärnan vinner visserligen aritmetikdelen, men eftersom det finns mycket mer styrkod skulle resultatet bli oavgjort.

STUDERAR MAN ETT DATABLAD är det lätt att dra slutsatsen att en viss MCU är vassast när det gäller låg energiförbrukning. Men datablad kan vara mycket missvisande även om det är sant att strömförbrukningen i vila och aktivt läge gynnar vissa MCU:er.

Energiförbrukningen kommer alltid att domineras av duty cycle (då arbete utförs). Om inte båda enheterna har samma duty cycle så är en jämförelse av specen i databladen i princip meningslös. Den arkitektur som passar en tillämpning bäst kommer i allmänhet också att ha lägst energiförbrukning.

Även periferifunktioner kan påverka effektförbrukningen på olika sätt. Till exempel har de flesta av Silicon Labs EFM32 32-bitars MCU:er en lågenergi-UART (LEUART) som kan ta emot data i lågeffektläge, medan endast två av företagets EFM8 8-bitars MCU:er erbjuder samma funktion. Alla tillämpningar som måste vänta på UART-trafik gynnas av denna funktion. Tyvärr finns det inget enkelt sätt att undersöka hur periferienheter påverkar just din tillämpning, utan det enda är att fråga din MCU-leverantör. Systemkonstruktören bör också vara ta reda på vilka energilägen som går att använda för olika uppgifter.

OM DU NU TAGIT HÄNSYN till allt jag skrivit ovan och det fortfarande inte är klart vilken MCU-arkitektur som passar bäst – vad göra? Grattis! Det betyder ju att båda är ett bra alternativ, och det inte spelar någon roll vilken arkitektur du använder. Istället kan du låta tidigare erfarenheter och personliga preferenser avgöra valet. Likaså kan det vara lämpligt att se framåt. Om de flesta framtida projekt tycks vara lämpade för ARM, välj då ARM, om framtidsprojekten däremot är mer fokuserade på små och billiga lösningar, välj 8051. ■