ELEKTRONIK TIDNINGEN

EXPERTARTIKEL

Anders Holmberg
Produktchef visualSTATE
IAR Systems

Så programmeras TI:s armbandsur

Anders Holmberg tar sig en titt på Texas Instruments utvecklarkit för klockan eZ430-Chronos och hittar en utmärkt plattform för att utforska grunderna i att designa tillståndsmaskiner

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

EMBEDDED EXPERT

Så programmeras TI:s armbandsur

IAR demonstrerar tillståndsmaskiner på TI Chronos

Av Anders Holmberg, IAR Systems AB

Efter examen arbetade Anders Holmberg en tid på Uppsala Universitet där han undervisade i datavetenskap och forskade om vetenskapliga och parallella beräkningar. År 2000 anställdes han av IAR Systems efter att däremellan ha arbetat på TietoEnator som som konsult och underleverantör inom telekommunikation och smartmobiler, teknisk programvara och inbyggda system.

TI:s innovativa kit eZ430-Chronos är baserat på en traditionell sportklocka med pulsmätning, en treaxlad accelerometer och en trycksensor, inneslutna i en attraktiv boett. Eftersom den kan kommunicera trådlöst kan klockan användas inte bara som en mångsidig sportklocka utan även som en trådlös datormus eller fjärrkontroll vid bildspel.

Om man tittar på en typisk sportklocka som en abstraktion, är den ett väldigt gott exempel på ett händelsedrivet system som vid varje given tidpunkt befinner sig i ett av flera skilda lägen eller underordnade lägen.

Exempelvis kan vi snabbt räkna upp ett antal abstrakta händelser som kan vara intressanta för ett program som byggas på den här plattformen:

- *Knapptryckningar*: Klockan har fem knappar. Det ursprungliga styrprogrammet reagerar på alla dessa och tilldelar även långa (två sekunder) tryckningar en betydelse för två av knapparna. Det kan också detektera och generera repetition — det vill säga upprepade knapptryckningar — som ger upphov till en rad knapphändelser.
- *Timerhändelser*: Ett antal timers kan köra och generera regelbundna händelser. För ett stoppur är det till exempel förmodligen viktigt med exakta timerhändelser i ganska hög hastighet för att hålla teckenfönstret uppdaterat. Pulsen avläses också regelbundet, åtminstone på programnivå – exakt hur kommunikationen med bröstbältet fungerar ligger dolt i en uppsättning proprietära bibliotek. Att användardefinierade alarm utlöses eller användardefinierade tröskelvärden för puls överskrids, et cetera, kan ses som abstrakta händelser som borde påverka hur programmet beter sig.

Klockan har ett antal olika lägen, eller tillstånd, där ovanstående händelser kan få olika innebörd. Exempelvis:

- Stoppur
- Inställning av datum och tid
- Aktivt radioläge för pulsmätning
- Väckarklocka
- Olika lägen för att avgöra vad som ska visas på skärmen
- Uppskattning av energiförbränning

Programvaran för en relativt begränsad styrkrets är ofta baserad på principen ”fat interrupt routine”, där det mesta av programlogiken är fördelad på ett antal avbrottsfunktioner. Till exempel är avbrottsfunktionen för huvudtimern för eZ430-Chronos ursprungliga styrprogram närmare 180 rader lång — enradskommentarer och blankrader inräknade. Den här funktionen styr mycket av programmets beteende, med hjälp av funktionsanrop och manipulering av globala data.

Detta tillvägagångssätt medför en rad fördelar. Det blir till exempel väldigt enkelt att styra de olika strömsparlägena hos klockans MSP430-processor, för det enda som huvudloopen behöver göra är att klara av lite lågnivåbearbetning innan den återvänder till rätt strömsparläge.

Riskmomentet är att programlogiken är utspridd över ett antal olika moduler, så det kan bli svårt att spåra hur olika egenskaper hos klockan interagerar vid debuggning eller när fler finesser läggs till.

Ett bättre tillstånd

Ett alternativt sätt att analysera ett sådant här program är att organisera det runt en huvudloop där det mesta av programlogiken har tagits om hand, och att förenkla avbrottsrutinerna så långt som möjligt så att de bara detekterar och rapporterar händelser till programmet. I ett tillstånds- och händelseorienterat program kan man sedan använda tillståndsmaskiner för att utforma och implementera logiken.

Fördelen är att det går att åstadkomma en prydlig åtskillnad mellan drivrutiner för indata och programlogik, och drivrutiner för utdata. Koden för att detektera indata och hantera utdata kan då enkelt

Figur 1. eZ430-Chronos är ett trådlöst utvecklingssystem för Texas Instruments 1 GHz rf-systemkrets TI CC430. Den har en LCD-display på 96 segment, trycksensor och treaxlig accelerometer.

återanvändas i olika projekt på samma hårdvara utan den behöver rensas på programspecifikt innehåll.

Nackdelen är att den bokföring som krävs för en centraliserad tillståndsmaskin kan bli väldigt otymplig allteftersom programkomplexiteten tilltar, åtminstone om programmet skrivs helt för hand utan stöd av högnivåverktyg. Vad man behöver är någon form av verktygsstöd. Med verktyg som till exempel vårt eget visualSTATE kan man utveckla ett händelsedrivet program baserat på tillstånd. Genom detta uppnår man enligt oss en rad fördelar:

- *Ökad produktivitet*. Naturligtvis finns det en inlärningskurva och man måste lägga sig till med några nya vanor, men efter ett tag kommer produktiviteten att öka eftersom man kan ägna allt mindre tid åt bokföring och fokusera mer på hur programmet fungerar. VisualSTATE genererar koden för tillståndsmaskinen, så du börjar direkt tjäna på att använda verktyget.
- *Ökad kvalitet*. Att använda tillståndsmaskiner räknas som en semiformell metod och genom att behandla ett tillstånds- och händelseorienterat problem som ett tillståndsmaskinsproblem redan i designfasen, kan man dra fördel av att arbeta i den mer formaliserade miljö som den väldefinierade semantiken hos tillståndsmaskiner erbjuder. Man kan börja simulera modellens beteende omedelbart för att försäkra sig om att den uppför sig som avsett. Man kan också använda formell verifiering för att säkerställa att det inte finns några baklås i designen eller onåbara tillstånd, et cetera.
- *En tydlig åtskillnad* mellan programlogiken och programmets in- och utdata (huvudsakligen den kortspecifika koden). Detta förenklar programarkitekturen och gynnar återanvändning mellan olika hårdvaruplattformar.

Som exempel (se figur 2) kan vi titta på en liten modell som designats för ez430-Chronos för att motverka knappstuds och avgöra om en knapptryckning är kort eller lång. I det här exemplet skickar den avbildade tillståndsmaskinen signaler till programdelen av tillståndsmaskinen för att tala om vilket slags knapptryckning den just avkodade.

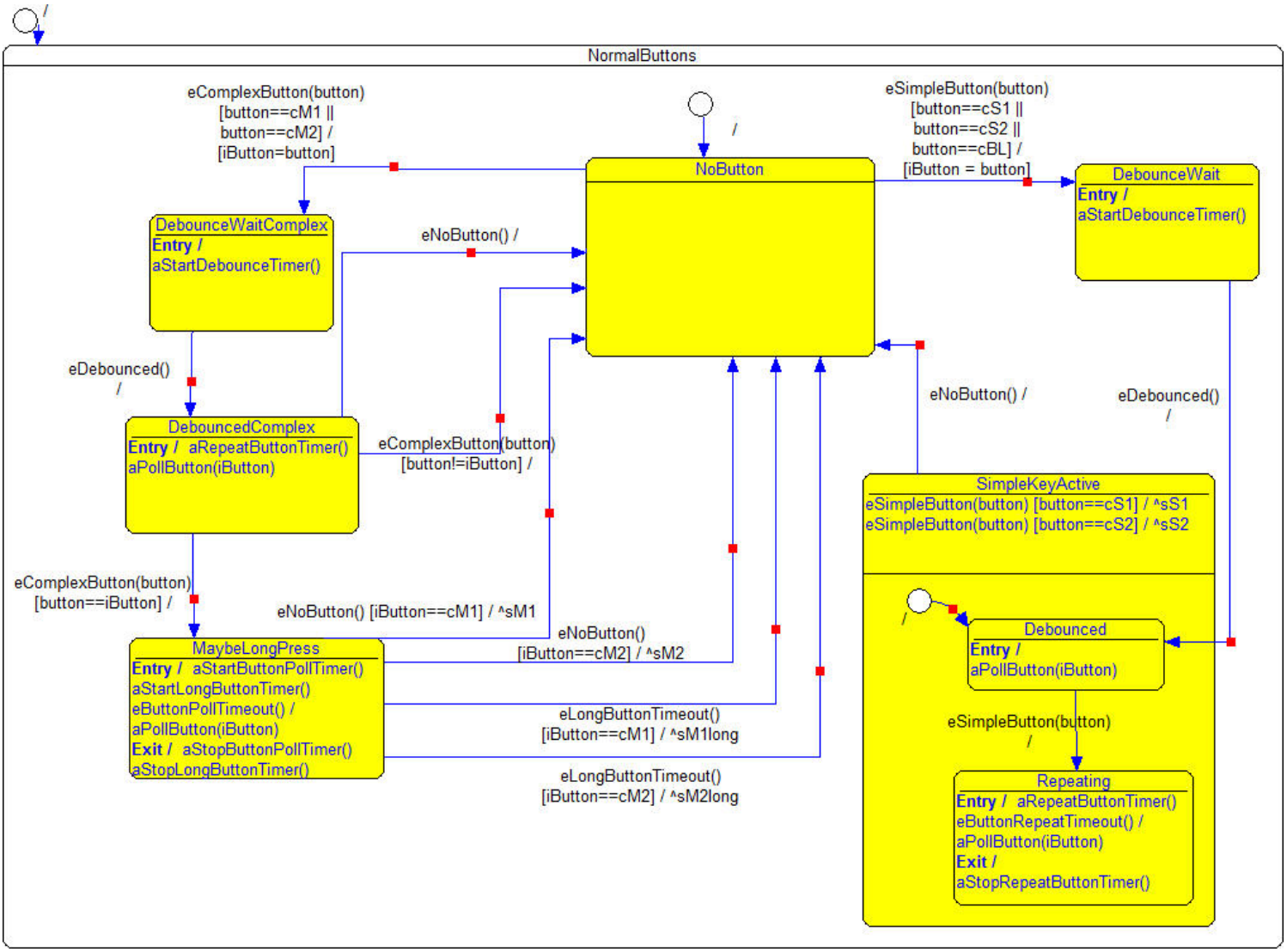
NoButton är det första tillståndet för den här tillståndsmaskinen, vilket man kan se på den lilla cirkeln (starttillståndet) och den följande övergången till tillståndet.

I ett riktigt produktionsfärdigt program vore kanske strategin att lägga logiken för knappstudshanteringen inuti tillståndsmaskinen inte det bästa alternativet eller ens ett alternativ som man överhuvud taget skulle komma på, men

ORDFÖRKLARING

Tillståndsmaskin

Tillståndsmaskiner används ofta för att lösa vissa typer av reglerproblem där flödesschemat kan avbildas som om det förflyttade sig via ett antal distinkta tillstånd. Det finns många olika definitioner och praktiska beskrivningar av vad en tillståndsmaskin är. I teorin är en tillståndsmaskin en sorts automat som svarar på en viss typ av indata genom att byta tillstånd.



Figur 2. En tillståndsmaskin för eZ430-Chronos för att motverka knappstuds och för att skilja mellan lång och kort tryckning.

det kan ändå vara ett bra exempel på hur kraftfulla hierarkiska tillståndsmaskiner är.

I den ursprungliga programkoden för klockan är den här funktionen fördelad på ett antal moduler och huvudlogiken för avkodningen och studshanteringen är en komplicerad switch-sats som breder ut sig över cirka 200 rader kod. Visst, den innehåller även högnivåprogramlogik, men det understryker bara ännu tydligare hur komplex den är.

Tillståndsmaskinen för att motverka knappstuds kan reagera på tre händelser: eComplexButton, eSimpleButton och eNobutton. Nu avgör helt enkelt avbrottsrutinen som reagerar på knapphändelser om avbrottet kommer från en av de knappar som inte gör någon skillnad på korta och långa knapptryckningar. Om så är fallet genereras en eSimpleButton-händelse. En eComplexButton-händelse genereras för de två knappar som tolkar korta och långa knapptryckningar som skilda indata. Tillståndsmaskinen kan i sin tur använda hjälpfunktionen aPollButton() för att omedelbart avgöra huruvida en viss knapp fortfarande är aktiv. Utöver de övriga knapphändelserna kan den här funktionen generera eNoButton-händelsen för att indikera att en knapp som nyss trycktes ner inte längre är aktiv.

Precis som den ursprungliga koden använder tillståndsmaskinen en uppsättning timers för att motverka knappstuds och avgöra längden på knapptryckningarna. När en knapptryckning detekteras rapporterars det till resten av tillståndsmaskinen (visas inte här) med hjälp av en signal – notationen syns exempelvis i övergångarna från tillståndet MaybeLongPress till NoButton (^sM1long). I enlighet med signalernas semantik bearbetas signalen omedelbart efter att tillståndsmaskinen har bearbetat händelsen som ledde till att signalen alstrades. ■

Mer information om hur du utformar en tillståndsmaskin för ditt program hittar du här: http://www.iar.com/vs/.