De geschiedenis herhaalt zich telkens weer:

Het moet goedkoper! En het kan veel beter!

Van een RTOS naar embedded Linux of Windows

Nog niet zo lang geleden was het hart van embedded software, 'het OS', altijd maatwerk. Maar zoals overal is ook daar een trend naar standaardproducten te zien. Zo zien we steeds vaker 'Windows' en 'Linux' verschijnen. Is dat een hype? Of is het een logische stap in de alsmaar voortgaande ontwikkeling om kosten te sparen en betere producten te maken? En wat betekent dat voor de toekomst?

Inhoud

Van ooit naar nu ...

Er is een duidelijke opkomst van generieke besturingssystemen, zoals Linux en Windows, in moderne elektronica. Ook veel 'realtime systemen' zullen de effecten gaan zien. Sommige zien dit als een ongewenste ontwikkeling, andere als één die niet te vermijden is. Echter, als we naar de geschiedenis kijken dan blijkt dat specifieke oplossingen telkens weer vervangen worden door goedkopere generieke onderdelen. Waarbij de generieke delen steeds krachtiger en omvangrijker worden.
In dit artikel blikken we terug op de geschiedenis van embedded systemen, zodat we de huidige opkomst van Linux en Windows kunnen verklaren en wellicht al een beetje in de toekomst kunnen kijken.

Van elektronica tot firmware

Al ruim voordat we kunnen spreken van 'embedded software' bestaan er al dergelijke systemen. Die producten werden opgebouwd uit discrete, vaak analoge, elektronica. We spreken niet voor niets (nog steeds) van 'elektronisch producten'. Kijken we terug, dan zien we dat de gebruikte analoge elektronica stap voor stap vervangen wordt door krachtigere en vooral goedkopere, digitale componenten. Eerst door eenvoudige, logische poorten zoals de beroemde '74xx TTL' logica. In de jaren '70 was dat hightech en modern; de jongere generatie ontwikkelaars zal dat anders zien. Want de poorten werden al snel vervangen door moderne, krachtigere en vooral goedkopere componenten.
Een belangrijke drijfveer voor de modernisering is de kostprijs. Het is veel goedkoper om standaard ICs te gebruiken dan om alles zelf te maken. Zelfs als dat betekent dat je een beetje meer elektronica gebruikt dan strikt noodzakelijk. Bovendien was de kwaliteit beter te garanderen. Immers, standaard componenten hebben veel gebruikers dus is de kans op fouten kleiner. Niet alleen omdat die producten goed zijn, maar ook omdat er veel kennis is hoe ze juist te gebruiken.
Een volgende stap in deze eindeloze wedloop naar beter en goedkoper was het vervangen van specifieke logica door generieke, programeerbare componenten. Met deze opkomst van de 'wondere wereld van de micro-elektronica' zoals Chriet Titulaer het noemde, begon software een

voorzichtige rol te spelen. We spreken dan veelal van ‘firmware’. Nog steeds wordt de functionaliteit vooral bepaald door de elektronica. Het ingebakken programma is eenvoudig, wordt veelal gemaakt door een elektrotechnisch ingenieur en nog niet echt belangrijk.

De opkomst van de programmeur

Langzamerhand wordt steeds meer elektronica vervangen door software. Software heeft het imago van goedkoop en flexibel. En dus wordt de programmatuur in het product steeds omvangrijker en belangrijker. In de begintijd was het maken van ‘de firmware’ een klusje voor de elektronica-afdeling. Waarschijnlijk vaak één van de minst populaire klusjes. Ongetwijfeld zijn heel wat ontwikkelprojecten uitgelopen omdat het schrijven van de software toch complexer was dan gedacht: Hoewel software goedkoop is om te reproduceren, is het niet goedkoop om te ontwikkelen. Zeker niet omdat door het imago van flexibel te zijn, het heel gebruikelijk wordt om de specificaties van de software continu te veranderen. Door al deze gelijktijdige invloeden ontstaat een nieuw specialisme, dat van de embedded-programmeur.

Het prototype embedded-programmeur heeft een elektronica achtergrond, zodat hij de schema’s van zijn collega kan lezen. Hij moet, gelijktijdig met de te ontwikkelen ‘hardware’, de ‘software’ schrijven voor het te ontwikkelen product. De naamgeving van deze twee nieuwe disciplines, ‘hard-’ en ‘software’, lijkt ook de rolverdeling aan te geven. In de praktijk worden de voorkeuren van de hardware afdeling veel meer ingewilligd dan die van de nieuwe software afdeling. Immers het veranderen van hardware is duur en dat van software niet.
De softwareafdeling komt hierdoor soms in het verdomhoekje. Haast per definitie zijn zij als laatste klaar; elke verandering heeft effect op de software. En elke fout moet door de software opgelost worden. Ook als het eigenlijk een hardware-ontwerp-probleem is; want software is goedkoop en flexibel ....

Natuurlijk heeft dat alles effect op het vak van embedded-programmeur. Steeds vaker gebruikt hij tools en halffabrikaten om de gevraagde kwaliteit en kwantiteit te leveren. En hij gebruikt een aanpak om de software zo flexibel mogelijk te maken. Dat begon door constanten expliciet te maken in headerfiles, maar ook door functionaliteit en implementatie te ontkoppelen door middel van een ‘API’.
Natuurlijk leert de embedded-programmeur ook van zijn collega’s in de bedrijfsprocesautomatisering. Moderne software-ontwikkelmethoden als ‘XP’ (Extreme Programming) zijn daar bedacht. Dat zijn methodes die er van uitgaan dat de specificaties continu veranderen. Als effect wordt ook embedded software steeds vaker incrementeel en iteratief ontwikkeld. Een aanpak die totaal anders is dan men gewend was bij het ontwikkelen van elektronica. De nieuwe aanpak zorgt ervoor dat de totale ontwikkeltijd korter wordt. En dus de kostprijs weer omlaag kan.
Inmiddels is ‘embedded software’ steeds omvangrijker geworden. Steeds meer functionaliteit wordt niet met elektronica, maar door software gerealiseerd. Die software is dus steeds groter, complexer en abstracter, en vooral steeds belangrijker geworden. Het is al lang niet meer geschikt om ‘het er even bij te doen’; programmeren is een vak geworden!

Het moet opnieuw goedkoper

Hoewel er soms negatief gesproken wordt over embedded software ontwikkeling –veel gehoorde klachten zijn ‘te duur’ en ‘te laat’– is er heel wat bereikt: de ‘elektronica’ is veel goedkoper geworden. Bovendien kan er veel meer en is de kwaliteit veel beter geworden.
Kijk maar om je heen: van TV tot GSM, de prijs/prestatie verhouding is gigantisch gegroeid. De omvang van de ingebedde software ook: er zit meer software in een stukje speelgoed dan in de ‘hoofdcomputer’ van een paar jaar terug!
Toch kunnen we niet op onze lauweren rusten. Vooral de embedded-programmeur niet. Want opnieuw zal de kostprijs lager moeten worden. En als het waar is dat bijna alle elektronica vervangen is door software, dan zal dit keer de software goedkoper moeten. Veel goedkoper! De

methode waarop dit zal gebeuren, ligt voor de hand. Dat wat in het verleden telkens opnieuw speelde, zal weer gebeuren. Maar nu met de software: Stap voor stap zal maatwerk steeds vervangen worden door een standaardproduct!

De introductie van het RT-OS

De eerste stappen op gebied van standaardisering zijn al gebeurd. Kijken we terug naar de trends op gebied van embedded systemen, dan zien we dat eerste systemen helemaal maatwerk waren. Als er al een besturingssysteem was, dan was dat typisch een eigen product. Vaak historisch (sommige mensen gebruiken de term 'hysterisch') gegroeid. En vaak niet meer dan een bibliotheek van handige routines, die moeilijk te gebruiken waren voor nieuwe programmeurs. Maar nuttig! Al was het maar omdat er lang niets beters was.
Dergelijke 'eigen besturingssystemen' hadden de neiging om te groeien. In omvang, maar ook in complexiteit en foutgevoeligheid. Ze werden dus steeds duurder om te onderhouden en te gebruiken. Vaak zijn ze daarom in de loop van de tijd vervangen door een meer-standaardproduct: een 'ingekocht' RT-OS, als VxWorks, pSOS, of QNX.
Als we kijken naar die traditionele embedded besturingsystemen, dan zijn ze allemaal ontstaan uit de behoefte aan een goedkoper en beter standaardproduct. Van de vele 'eigen' ontwikkelingen zijn de best bruikbare, de best gedocumenteerde en de best presterende besturingsystemen overgebleven.
Omdat er vaak maar weinig verschil was tussen 'embedded systemen' en 'realtime systemen' hadden embedded besturingsystemen die ook realtime waren een duidelijk voordeel: een grotere markt.

Hoe standaard is een standaard RT-OS?

Er zijn tientallen, wellicht zelfs honderden van dergelijke traditionele RT-OS'en. Hoewel er een paar duidelijke marktleiders zijn, kunnen we ons afvragen in hoeverre het standaardproducten betreft. Zo claimen sommige dat ze voldoen aan een standaard als 'POSIX', maar dat is vaak erg beperkt. Ze zijn daarom vaak moeilijk te gebruiken. Zo is de huidige generatie studenten gewend aan Windows en Linux.

Ook de hardware verandert

Het is typisch voor embedded software dat elke verandering in de hardware effecten heeft op de software. Dus als we de huidige ontwikkelingen in de embedded besturingssystemen willen begrijpen, zullen we moeten kijken wat de hardware doet.
Hoewel de softwareafdeling vaak negatieve gevoelens heeft bij hardwareveranderingen, is dat niet nodig als we naar de algemene trend kijken. Want hoewel er steeds meer functionaliteit verschuift van elektronica naar software, mag die software gebruik maken van steeds krachtigere standaardcomponenten.
In de ratrace naar steeds maar goedkoper, zijn de standaardcomponenten die te koop zijn ook steeds beter en krachtiger geworden. Een goed voorbeeld is geheugen. Hoe lang is het geleden dat embedded programma's slechts een paar kilobyte RAM mochten gebruiken? Inmiddels zijn dergelijke kleine geheugens feitelijk niet meer te koop; een standaardmodule met meerdere Mbytes is vaak goedkoper.
Ook de processor wordt steeds sneller en krachtiger. Een paar decennia geleden was 8bit de norm, toen kwamen de 16bitters. Inmiddels zijn begrippen als '32bits', een floating point unit (FPU) en zelfs een hardware-MMU (Memory Management Unit) heel gebruikelijk geworden. Ook in embedded systemen!

Ook hardware maakt systemen 'RT' ...

Standaard hardware zorgt er ook voor dat allerlei realtime-eisen steeds gemakkelijker te realiseren zijn. Om uit te leggen wat realtime is, wordt vaak naar de seriële poort verwezen. Hierbij is het belangrijk om snel te reageren op een interrupt, anders zou er data verloren gaan.

Dit is nog steeds een goed voorbeeld. Want de huidige standaardcomponenten maken standaard gebruik van buffers. Die extra hardware slaat automatisch data tijdelijk op; totdat de software tijd heeft om meerdere berichten in één keer uit te lezen. Het snel genoeg reageren op een interrupt is dus veel makkelijker geworden.
Dit kunnen we ook zien aan een generiek besturingssysteem als 'Windows', dat op meer dan 90% van alle desktops gebruikt wordt en zeker niet realtime is. Toch hoor ik nooit klachten over het missen van interrupts. Terwijl de seriële poort typisch ingesteld is op 115Kbit/s als men surft over een analoog modem. Ook niet bij communicatie over de veel snellere USB verbinding en zelfs niet bij ethernet dat tegenwoordig een snelheid haalt van een Gigabit! Terwijl de oude 'realtime' berekeningen veelal uitgingen van maximaal 1200 baud!
Een vergaande standaardisering heeft elektronica –de hardware– steeds goedkoper gemaakt. Maar die standaard hardware heeft moderne systemen ook veel sneller en betrouwbaarder gemaakt. Zowel direct door rauwe computerkracht, als indirect door slimmere oplossingen.
Als uitdaging voor de lezer de vraag: hoeveel instructies kan een moderne CPU uitvoeren tussen twee interrupts? En als filosofische opmerking: hoe moeilijk is het dan, gezien dit gigantische aantal, om software niet realtime te laten reageren?

En dus moet de software beter en goedkoper

Samenvattend kunnen we zeggen dat 'elektronica' continu goedkoper en beter wordt en moet worden. Maar ook dat dit bereikt is door steeds meer maatwerk te verruilen voor standaardproducten en door de functionaliteit te verschuiven naar software. Als gevolg is de hardware sneller en goedkoper geworden, en de software steeds complexer.
Het is de vraag hoelang deze lijn nog voortgezet kan worden. Er zal altijd hardware nodig zijn om de software op te laten draaien en dus zal er een einde komen aan de besparingen op de hardware.
Dus zal er, om de kosten nog verder te reduceren, bespaard moeten worden op software. Er lijkt hier veel ruimte om op dezelfde manier kosten te besparen: Net als bij de hardware, zal embedded software steeds minder maatwerk en steeds meer op standaardproducten gebaseerd worden. Het is nauwelijks een vraag of de eerste poging hierin, het inzetten van het traditionele 'standaard' RT-OS, zal volstaan.

Wat is er dan mis met mijn RT-OS?

Het traditionele RT-OS was jaren geleden een prima oplossing om de kosten omlaag te brengen. En voor sommige omgevingen zijn er wellicht geen betere alternatieven te vinden. De vraag is echter of het nog steeds de beste keuze is?
Ten eerste zijn die RT-OS producten nog steeds erg specialistisch en veelal gebaseerd op verouderde hardware; toen FPU's en MMU's niet bekend waren. Ook de omgang met megabytes aan geheugen is vaak complex. Kunt u bijvoorbeeld opgeven in welke delen van het RAM niet geschreven mag worden? Moderne CPUs kunnen dat afdwingen en controleren. Zo nee, dan mist u een mogelijkheid om fouten te detecteren.
Tegelijkertijd zijn traditionele RT-OS'en eenkennig: niet elk embedded product is ook realtime. Toch gaan de meeste RT-OS'en ervan uit dat u hard-realtime eisen heeft, terwijl dat vaak de duurste eisen zijn. Daar moeten kosten te besparen zijn. En dan spreken we nog niet eens van moderne eisen als 'mobility', 'networking' en 'direct-on'. Wordt dat ondersteund, of zit het in de weg?

Kan het goedkoper?

Gezien de hoeveelheid maatwerk, de omvang en de geringe standaardisering van embedded software mogen we verwachten dat veel softwareprojecten minimaal de opdracht krijgen om behoorlijk bij te dragen op het besparen van de kosten.
Terecht of niet, standaardproducten als 'Linux' of 'Windows' worden steeds vaker gebruikt in embedded systemen. Zeker is dat ze, zeker voor de jonge, verwende programmeur eenvoudig te gebruiken zijn. En dat de licentiekosten van deze producten verleidelijk zijn.

Wie gaat het doen?

Deze luchtige terugblik op de geschiedenis van embedded systemen laat zien dat de kosten telkens weer omlaag moesten en de kwaliteit omhoog. En dat dit veelal gebeurt door steeds meer standaard elektronica te gebruiken en steeds meer software. Telkens weer bleek maatwerk te duur. Producten die niet meegingen in deze vooruitgang, zijn verdwenen. Voor de iets oudere lezer is die 74XX serie een goed voorbeeld.
Het kan niet anders, dan dat de geschiedenis zich herhaalt.

En nu software het belangrijkste onderdeel van ‘elektronica’ is geworden, de softwarelicenties soms de helft van de Bill-of-Materials vormen en de softwareafdeling vaak de grootste ontwikkelafdeling is, kan het niet anders dat de software nu aan de beurt is.
Ook embedded software zal goedkoper en beter moeten. En dat kan door standaardproducten te gebruiken. Dat kan embedded Windows zijn, dat substantieel goedkoper is dan veel traditionele RTOS’en. Of het kan Linux worden, dat als OpenSource product licentievrij is. Of dat kan een derde zijn.
De geschiedenis leert ons niet waarmee we kunnen winnen, alleen dat we moeten innoveren en dat als wij niet op de kosten letten, de concurrent het wel zal doen!
Onze software kosten moeten dus omlaag. Maar dan moeten we weten waarom de kosten hoger zijn dan vaak gedacht. Lees dat bijvoorbeeld in het artikel ‘*Hoe kan gratis Linux toch te duur zijn’”*.

ALbert Mietus