***INTERRUPT***

1. **Tujuan**

* Memahami kebutuhan untuk melakukan banyak tugas secara bersamaan
* Memahami ukuran kinerja *real-time* seperti *bandwidth* dan *latency*
* Mempelajari bagaimana *interrupt* dapat digunakan untuk memperkecil *latency*
* Mempelajari dasar-dasar pemrograman *interrupt*: *arm*, *enable*, *trigger*, *vector*, *priority*, *acknowledge*
* Memahami bagaimana menggunakan SysTick untuk membuat *interrupt* periodik
* Menggunakan SysTick untuk membuat bunyi

1. **Peralatan yang digunakan**

* Keil μVision v5
* Tiva C Series LaunchPad
* Stellaris® ICDI Drivers
* Resistor 1,5 kΩ
* *Speaker*

1. **Landasan teori**

**Interrupt** adalah transfer otomatis eksekusi perangkat lunak dalam menanggapi kejadian perangkat keras yang asinkron dengan eksekusi perangkat lunak. Kejadian perangkat keras ini disebut **trigger**. Kejadian perangkat keras dapat berupa transisi sibuk ke siap dalam perangkat I/O eksternal (seperti input/output UART) atau kejadian internal (seperti bus fault, memory fault, atau periodic timer). Ketika perangkat keras membutuhkan layanan, ditandai dengan transisi keadaan sibuk ke siap, perangkat keras akan meminta interrupt dengan mengatur bendera trigger-nya. **Thread** didefinisikan sebagai lintasan aksi perangkat lunak ketika dijalankan. Eksekusi rutin layanan interrupt disebut thread latar belakang. Thread ini diciptakan oleh permintaan interrupt perangkat keras dan dibunuh ketika rutin layanan interrupt kembali dari interrupt (misal dengan perintah **BX LR**). Thread baru dibuat untuk masing-masing permintaan interrupt. Penting untuk dipertimbangkan ketika masing-masing permintaan individu sebagai thread terpisah karena variabel lokal dan register digunakan dalam rutin layanan interrupt adalah unik dan terpisah dari satu kejadian interrupt ke interrupt berikutnya. Dalam sistem **multi-threaded**, kita menganggap thread bekerja sama untuk melakukan keseluruhan tugas. Akibatnya kita akan mengembangkan cara untuk thread berkomunikasi (misal FIFO) dan melakukan sinkronisasi dengan satu sama lain. Kebanyakan sistem embedded memiliki tujuan tunggal. Di sisi lain, komputer serba guna dapat memiliki banyak fungsi yang tidak terkait untuk tampil. **Proses** juga didefinisikan sebagai aksi perangkat lunak ketika dijalankan. Proses tidak selalu bekerja sama menuju tujuan umum. Thread membagi akses ke perangkat I/O, sumber daya sistem, dan variabel global, sedangkan proses memiliki variabel global dan sumber daya sistem yang terpisah. Proses tidak membagi perangkat I/O.

Tidak ada definisi standar untuk istilah mask, enable, dan arm dalam komunitas Teknik Komputer. **Arm** suatu perangkat berarti memungkinkan trigger perangkat keras ke interrupt. Sebaliknya, **disarm** suatu perangkat berarti mematikan atau memutuskan trigger perangkat keras dari interrupt. Setiap trigger interupsi potensial memiliki bit arm yang terpisah. Sesuatu melakukan arm suatu trigger jika sesuatu itu tertarik untuk interrupt dari sumber. Sebaliknya, sesuatu melakukan disarm suatu trigger jika sesuatu itu tidak tertarik untuk interrupt dari sumber. **Enable** berarti memungkinkan interrupt saat ini. Sebaliknya, **disable** berarti menunda interrupt sampai lain waktu. Pada prosesor ARM Cortex-M, ada satu bit enable interrupt untuk keseluruhan sistem interrupt. Kita disable interrupt jika saat ini tidak sesuai untuk menerima interrupt. Pada khususnya, untuk disable interrupt, kita mengatur bit I dalam **PRIMASK**. Dalam bahasa C, kita enable dan disable interrupt dengan memanggil fungsi **EnableInterrupts()** dan **DisableInterrupts()**.

Perangkat lunak memiliki kontrol dinamis atas beberapa aspek urutan permintaan interrupt. Pertama, setiap trigger interrupt potensial memiliki bit **arm** terpisah yang perangkat lunak dapat aktifkan atau nonaktifkan. Perangkat lunak akan mengatur bit arm untuk perangkat dari mana perangkat itu ingin menerima interrupt, dan menonaktifkan bit arm dalam perangkat dari mana interrupt tidak diizinkan. Dalam kata lain, perangkat lunak menggunakan bit arm untuk memilih perangkat yang akan dan perangkat yang tidak akan meminta interrupt. Untuk kebanyakan perangkat, ada bit enable dalam NVIC yang harus diatur (interrupt SysTick periodik adalah pengecualian, tidak memiliki enable NVIC). Aspek ketiga yang perangkat lunak kontrol adalah bit enable interrupt. Pada khususnya, bit 0 register khusus **PRIMASK** adalah bit mask interrupt, **I**. Jika bit ini adalah 1, kebanyakan interrupt dan eksepsi tidak diperbolehkan, yang didefinisikan sebagai **disabled**. Jika bit adalah 0, maka interrupt diperbolehkan, yang didefinisikan sebagai **enabled**. Aspek keempat adalah prioritas. Register **BASEPRI** mencegah interrupt dengan interrupt prioritas lebih rendah, tetapi memungkinkan interrupt prioritas lebih tinggi. Misalnya jika perangkat lunak mengatur **BASEPRI** ke 3, maka permintaan dengan level 0, 1, dan 2 dapat interrupt, sedangkan permintaan pada level 3 dan lebih tinggi akan ditunda. Perangkat lunak juga dapat menentukan level prioritas setiap permintaan interrupt. Jika **BASEPRI** adalah nol, maka fitur prioritas disabled dan seluruh interrupt diperbolehkan. Aspek kelima adalah trigger perangkat keras eksternal. Salah satu contoh trigger perangkat keras adalah bendera **Count** dalam register **NVIC\_ST\_CTRL\_R** yang diatur secara periodik oleh SysTick. Contoh lain trigger perangkat keras adalah bit dalam register **GPIO\_PORTF\_RIS\_R** yang ditetapkan pada tepi naik atau turun pin input digital. Lima kondisi harus benar untuk interrupt yang akan dihasilkan:

1. Arm perangkat
2. Enable NVIC
3. Enable global
4. Level prioritas interrupt harus lebih tinggi daripada level eksekusi saat ini
5. Trigger kejadian perangkat keras

Agar interrupt dapat terjadi, lima kondisi di atas harus benar namun dapat terjadi dalam urutan apapun.

Interrupt menyebabkan urutan lima kejadian berikut. Pertama, instruksi saat ini selesai. Kedua, eksekusi program yang sedang berjalan ditunda, mendorong delapan register ke stack (**R0**, **R1**, **R2**, **R3**, **R12**, **LR**, **PC**, dan **PSR** dengan **R0** di atas). Jika unit floating point pada TM4C123 aktif, 18 word tambahan akan didorong ke stack mewakili keadaan floating point, sehingga total 26 word. Ketiga, **LR** diatur ke nilai tertentu menandakan interrupt service routine (ISR) sedang dijalankan (bit [31:4] ke 0xFFFFFFF, dan bit [3:0] menentukan jenis interrupt yang kembali tampil). Sebagai contoh, kita akan melihat LR diatur ke 0xFFFFFFF9. Jika register floating point didorong, LR akan menjadi 0xFFFFFFE9. Keempat, **IPSR** diatur ke nomor interrupt yang diproses. Akhirnya, **PC** dimuat dengan alamat ISR (vektor).

1. Instruksi saat ini selesai
2. Delapan register didorong ke stack
3. LR diatur ke 0xFFFFFFF9
4. IPSR diatur ke nomor interrupt
5. PC dimuat dengan vektor interrupt

Lima langkah di atas, disebut **context switch**, terjadi secara otomatis dalam perangkat keras seperti konteks diaktifkan dari thread latar depan ke thread latar belakang. Kita juga dapat memiliki context switch dari ISR prioritas lebih rendah ke ISR prioritas lebih tinggi. Berikutnya, perangkat lunak mengeksekusi ISR.

Jika bendera trigger diatur, tetapi interrupt disabled (I=1), level interrupt tidak cukup tinggi, atau bendera disarmed, permintaan tidak dihentikan. Melainkan permintaan menjadi **pending**, ditunda sampai lain waktu, ketika sistem menganggap nyaman untuk menangani permintaan. Dalam kata lain, sekali bendera trigger diatur, dalam sebagian besar kasus bendera itu tetap sampai perangkat lunak membersihkannya. Lima kejadian penting (arm perangkat, enable NVIC, enable global, level, dan trigger) dapat terjadi dalam urutan apapun. Sebagai contoh, perangkat lunak dapat mengatur bit I untuk mencegah interrupt, menjalankan beberapa kode yang perlu dijalankan sampai selesai, kemudian membersihkan bit I. Trigger terjadi saat dijalankan dengan I=1 ditunda sampai waktu bit I dibersihkan lagi.

Pembersihan bendera trigger disebut **acknowledgement**, yang hanya terjadi oleh aksi perangkat lunak khusus. Setiap bendera trigger memiliki perangkat lunak aksi khusus yang harus dilakukan untuk membersihkan bendera itu. Kita akan memberikan perhatian khusus pada aksi perangkat lunak enable/disable ini. Interrupt periodik SysTick akan menjadi satu-satunya contoh acknowledgement otomatis. Untuk SysTick, timer periodik meminta interrupt, tetapi bendera trigger secara otomatis akan dihapus ketika ISR dijalankan. Untuk semua bendera trigger lainnya, ISR secara eksplisit harus mengeksekusi kode yang membersihkan bendera.

**Interrupt service routine** (ISR) adalah modul perangkat lunak yang dieksekusi ketika perangkat keras meminta interrupt. Mungkin ada satu ISR besar yang menangani seluruh permintaan (interrupt disurvei), atau banyak ISR kecil khusus untuk setiap sumber potensial interrupt (interrupt vektor). Desain interrupt service routine memerlukan pertimbangan cermat pada banyak faktor. Kecuali untuk interrupt SysTick, perangkat lunak ISR secara eksplisit harus membersihkan bendera trigger yang menyebabkan interrupt (acknowledge). Setelah ISR menyediakan layanan yang diperlukan, ISR akan mengeksekusi **BX LR**. Karena LR mengandung nilai khusus (misal 0xFFFFFFF9), instruksi ini menarik 8 register dari stack, yang mengembalikan kontrol ke program utama. Jika LR adalah 0xFFFFFFE9, maka 26 register (R0-R3,R12,LR,PC,PSR, dan 18 register floating point) akan ditarik oleh **BX LR**. Ada dua stack pointer: PSP dan MSP. Sangat penting bahwa perangkat lunak ISR menyeimbangkan stack sebelum keluar. Eksekusi thread sebelumnya kemudian akan dilanjutkan dengan nilai stack dan register yang tepat yang ada sebelum interrupt. Meskipun pemegang interrupt dapat menciptakan dan menggunakan variabel lokal, parameter yang lewat antar thread harus dilaksanakan menggunakan variabel shared global memory. Variabel global private dapat digunakan jika thread interrupt ingin menyampaikan informasi ke dirinya sendiri, misal dari satu interrupt ke yang lain. Eksekusi program utama disebut thread latar depan, dan eksekusi berbagai interrupt service routine disebut thread latar belakang.

Aksioma dengan sinkronisasi interrupt adalah ISR harus mengeksekusi secepat mungkin. Interrupt harus terjadi ketika saatnya untuk melakukan fungsi yang dibutuhkan, dan interrupt service routine harus melakukan fungsi itu, dan kembali segera. Menempatkan percabangan mundur (loop busy-wait, iterasi) dalam perangkat lunak interrupt harus dihindari jika mungkin. Persentase waktu yang dihabiskan untuk mengeksekusi perangkat lunak interrupt harus kecil bila dibandingkan dengan waktu antara trigger interrupt.

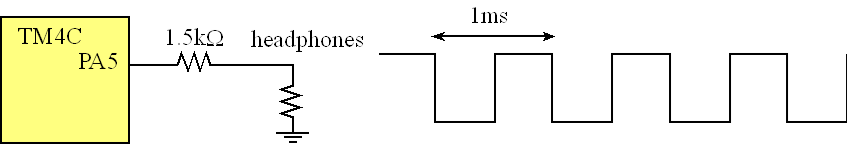
1. **Eksperimen**

Desain suatu antarmuka *speaker* 32 Ω dan gunakan untuk membangkitkan bunyi 1 kHz.

Untuk membuat bunyi, ciptakan gelombang persegi. Pada 3,3 V, *speaker* 32 Ω akan membutuhkan arus sekitar 100 mA. TM4C123 dapat menghasilkan arus output 8 mA. Jika ditempatkan resistor seri dengan headphone, maka arus hanya akan mA. Untuk membangkitkan bunyi 1 kHz dibutuhkan gelombang persegi 1 kHz. Ada banyak metode untuk membangkitkan gelombang persegi. Aktifkan interrupt periodik dan ubah pin output dalam ISR. Untuk membangkitkan gelombang 1 kHz ubah pin PA5 setiap 500 μs. Diasumsi PLL aktif dan sistem berjalan pada 80 MHz. Inisialisasi SysTick untuk interrupt dengan periode 500 μs. Nilai yang benar untuk reload adalah 39999 . Jika frekuensi bus 16 MHz, atur nilai reload 7999 . Karena output gelombang bunyi ini adalah sinyal real-time, atur prioritasnya ke level tertinggi, yaitu 0. Lihat Program 6.

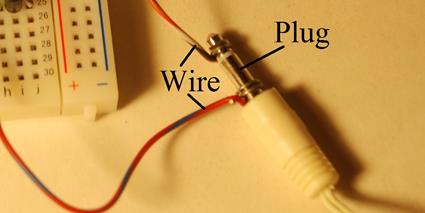
|  |
| --- |
| **void Sound\_Init(void){ unsigned long volatile delay;   SYSCTL\_RCGC2\_R |= 0x00000001; // activate port A   delay = SYSCTL\_RCGC2\_R;   GPIO\_PORTA\_AMSEL\_R &= ~0x20;      // no analog   GPIO\_PORTA\_PCTL\_R &= ~0x00F00000; // regular function   GPIO\_PORTA\_DIR\_R |= 0x20;     // make PA5 out   GPIO\_PORTA\_DR8R\_R |= 0x20;    // can drive up to 8mA out   GPIO\_PORTA\_AFSEL\_R &= ~0x20;  // disable alt funct on PA5   GPIO\_PORTA\_DEN\_R |= 0x20;     // enable digital I/O on PA5   NVIC\_ST\_CTRL\_R = 0;           // disable SysTick during setup   NVIC\_ST\_RELOAD\_R = 39999;     // reload value for 500us (assuming 80MHz)   NVIC\_ST\_CURRENT\_R = 0;        // any write to current clears it   NVIC\_SYS\_PRI3\_R = NVIC\_SYS\_PRI3\_R&0x00FFFFFF; // priority 0                  NVIC\_ST\_CTRL\_R = 0x00000007;  // enable with core clock and interrupts   EnableInterrupts(); }**  **void SysTick\_Handler(void){   GPIO\_PORTA\_DATA\_R ^= 0x20;     // toggle PA5 }** |

Program 6. Output bunyi menggunakan interrupt periodik (C12\_SoftSound)



Gambar 6. Output gelombang persegi terhubung ke speaker akan membangkitkan bunyi

Gambar 7 memperlihatkan sistem tanpa audio jack.



Gambar 7. Cara sederhana menghubungkan headphone ke rangkaian

Tanggalkan sekitar 3 cm salah satu ujung kabel, lilit bagian yang dilucuti pada kabel di sekitar audio plug headphone. Hati-hati agar dua kabel tidak terhubung singkat .

**Instruksi**:

* Buat rangkaian pada *protoboard* berdasarkan Gambar 6 dan Gambar 7.
* Jalankan C:\Keil\TExaSware\C12\_SoftSound\SoftSound.uvproj.
* Periksa kesesuaian Program 6 dengan SoftSound.uvproj.
* Gunakan simulator, amati apa yang terjadi.
* Gunakan Stellaris ICDI, amati apa yang terjadi.

1. **Bibliografi**

<http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C12_Interrupts.htm>