
Gömülü Sistemler 2

Contents

1 Haberleşme Protokollerİ	2
2 Seri Haberleşme Protokollerİ	2
2.1 UART	2
2.2 I2C	3
2.3 SPI (Serial Peripheral Interface)	4
2.4 Controller Area Network (CAN)	6
2.5 Karşılaştırma	7
3 ADC (Analog to Digital Converter)	8
3.1 Flash ADC	9
3.2 Sigma-Delta ADC	9
3.3 Successive Approximation (SAR)	10
4 Genel Amaçlı Zamanlayıcılar ve Motor Kontrolü	10
4.1 Clock Sinyali Oluşturma	10
4.2 Timer'lar	10
4.3 Bekçi Köpeği Zamanlayıcısı (Watchdog Timer)	11
4.4 PIC16F877 Timer Block Diagram	11
4.5 MSP430 Timer Block Diagram	12
4.6 Timer in ARM Microcontrollers	13
4.7 PWM Modu	14
4.8 Ultrasonik Mesafe Ölçümü	14
4.9 Motor Tipleri	15
4.9.1 Servo Motor	15
4.9.2 DC Motor	16
5 Gerçek Zamanlı Sistemler	16
5.1 Embedded Firmware	16
5.1.1 RTOS Temel Kavramlar	17
6 Scheduling	19
6.1 Task Specification	19
6.2 Yaygın Scheduling Yaklaşımları	20

1 Haberleşme Protokollerı

Motivation

- Connect different systems together
 - Two embedded systems
 - A desktop and a embedded system
- Connect different chips in the same embedded system
 - MCU to peripheral
 - MCU to MCU
- Without using a lot of I/O lines
 - I/O lines require I/O pads which costs \$\$\$ and size
 - I/O lines require PCB area which costs \$\$\$ and size
- Often at relatively low data rates
- But sometimes at higher data rates

2 Seri Haberleşme Protokollerı

Az sayıda I/O ve her bir çevrimde bir bit gönderim.

- Inter-integrated Circuit (I2C)
- Serial Peripheral Interface (SPI)
- Controller Area Network (CAN)
- Asenkron Seri Haberleşme (UART → RS232, RS485 ve RS422)
- USB

2.1 UART

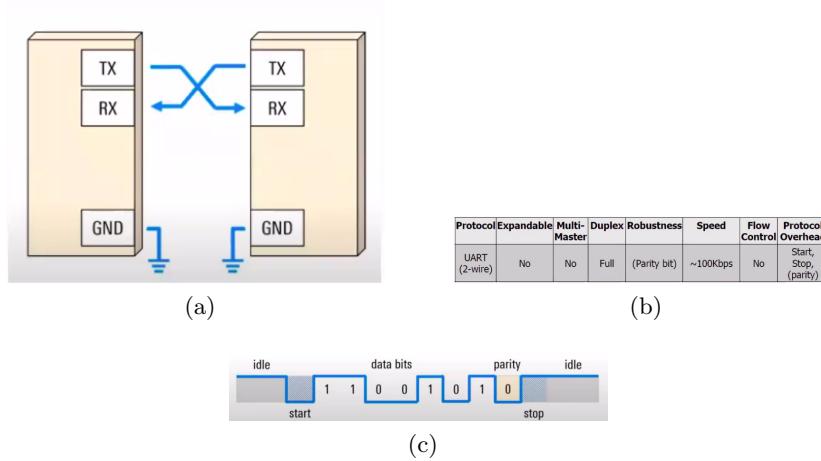


Figure 1: UART

- RX ve TX olmak üzere 2 kablolu hatta ve bir ground hattına ihtiyaç duyar.

- 1'e 1 haberleşme yapabilir (peer 2 peer).
- Asenkron haberleşmeye sahiptir.
- 15 metre uzaklığa kadar iletim uzunlupğu.
- Haberleşme biti konfigüre edilebilir.
- En eski protokollerden birisi olup RS232, COM portu, modemle kullanılmaktadır.
- Oldukça ucuz.
- UART bir protokoldür. Gerilim seviyelerinin ne olacağı RS-XXX standardı ile belirlenebilir.
- Her ne kadar işlem birimlerinde UART çıkışı olsa da sinyal seviyesinin terslenmesi ve belirli gerilim aralığına gerilimesi gerekmektedir (0V, 5V : 12V, -12V). Bu çözüm level shifter devreleri ile sağlanmaktadır.

2.2 I2C

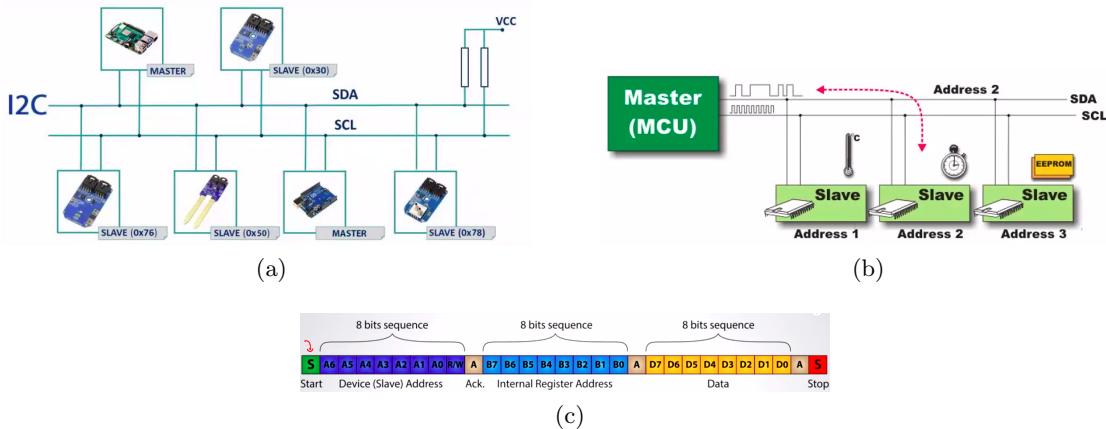


Figure 2: I2C

- Master-slave konfigürasyonuyla birden fazla aygıtlı kısa mesafeli senkron haberleşme yapabilmektedir.
- Genellikle bilgisayar aygıtlarının haberleşmesinde kullanılmazlar.
- Çoklu master yapısını destekleyebilir.
- SDL (Serial Data Line) ve SCL (Serial Clock Line) olmak üzere iki kablolu hatta sahiptir.
- Pull-up dirençleri en önemli parçasıdır.
- Her bir bit, bir clockta iletilmektedir.
- SDA üzerindeki veri clock periyodu boyunca stabil olmalıdır.
- Tüm slave'ler aynı iletim hattını paylaşır ve ilk byte'ı aynı zamanda alır. Fakat sadece bir tanesi bu adresle uyuşur.
- Başlama biti üretilir, slave adresi gönderilir, eğer bu adreste bir slave varsa acknowledge biti üretilir, slave'de ulaşmak istenilen register adresi gönderilir, bu istek geçerliyse slave cevap verir.
- Read: 1, write: 0.

2.3 SPI (Serial Peripheral Interface)

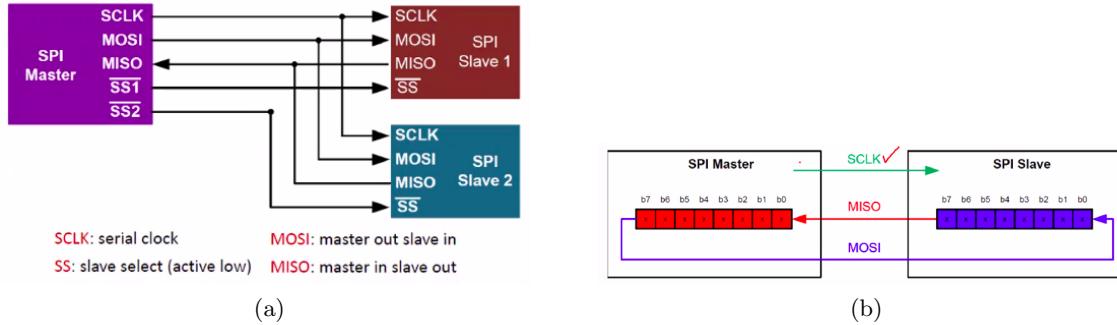
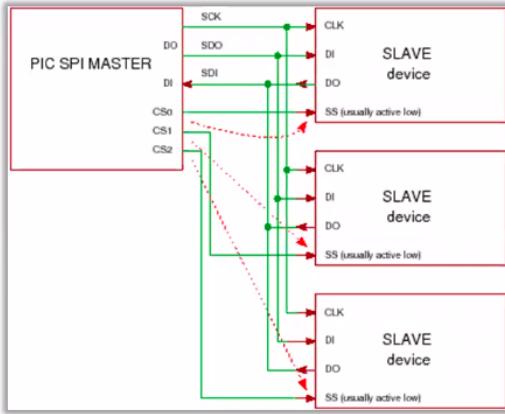
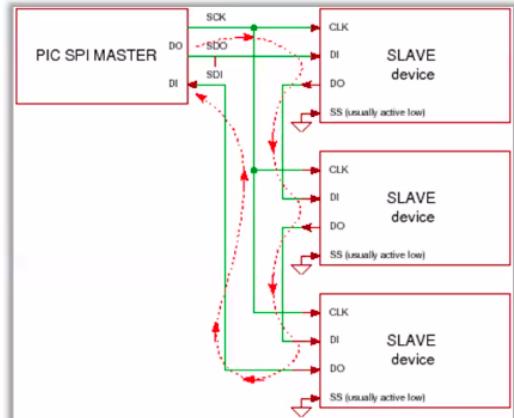


Figure 3: SPI

- Senkron, full-duplex, seri haberleşme.
- En az 4 I/O gereklidir.
- Tek master fakat birden fazla slave aygıta uyumludur.
- Herhangi bir ACK biti yok.
- Yukardakilerine göre daha hızlı (birkaç Mbps).
- ADC dönüştürücüler gibi data streamler için uygundur.
- Eğer tek bir slave varsa SS pini lojik sıfıra sabitlenir.
- Seri port üzerinden ses, video vb. göndermekte baya avantajlı.
- I2C'de her veri alışverişinde slave'in adresi gerekiyordu. Burada SS tek bitte bu olayı çözüyor.
- Pek çok sensör tarafından kullanılmaktadır.
- Dezavantaj: ACK biti yok. Çoklu aygit durumunda problem.
- Çalışma Mantığı:
 - SCLK konfigüre edilir. Clock frekansı iletişim hızını belirler.
 - SS ile ilgili aygıta bağlı pin lojik sıfıra çekilir.,
 - MISO ve MOSI eşzamanlı transfer yapar.
 - İşlem bittikten sonra SS tekrardan lojik 1 yapılır.
- Eşzamanlı senkron data exchange.
- İki farklı yapı olabilir:



(a) Master and Multiple Independent Slaves



(b) Master and Multiple Daisy-Chained Slaves

- a) daha az I/O port'u ama daha hızlı. b) daha az I/O portu ama daha yavaş.

2.4 Controller Area Network (CAN)

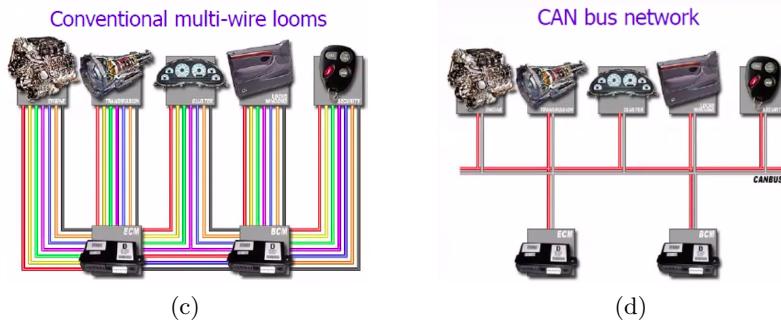


Figure 4: SPI

- Hızlı seri bus: güvenilir, verimli, ekonomik.
- 40 farklı device ile 1Mbit/s'ye kadar 40 metrelük bir alanda kullanılabilir.
- Otomobil, makine, fabrika etc.
- Adres tabanlı değil mesaj tabanlı: mesaj içinde tüm bilgiler bulunuyor ve bunu alan aygıtlar kendileri yorumluyor.
- Mesaj önceliği var.
- Bir veri bus'a aktarıldığında, bus üzerindeki birden fazla sistem bu bilgiyi alabilir.
- Herkes master.
- Veri paketleri 8 byte.



Figure 5: CAN Message

- **SOF:** Start of packet.
- **CAN-ID:** Message-ID –> Öncelik ve fonksiyonel adres
- **RTR:** 1 ise mesaj karşısından isteniyor, 0 ise mesaj veri içeriyor
- **Control:** Mesaj içeriğinin kaç byte olduğu.
- **CRC:** Cyclic Redundancy Check –> Double check
- **ACK:** Mesajı alan var mı yok mu?

2.5 Karşılaştırma

Protocol	Expandable	Multi-Master	Duplex	Robustness	Speed	Flow Control	Protocol Overhead
UART (2-wire)	No	No	Full	(Parity bit)	~100Kbps	No	Start,Stop, (parity)
I2C	Large	Yes	Half	ACK/NACK	S:100Kbps F:400Kbps Hs:3.4Mbps	Yes	Start, 7-bit addr, R/W, ACK/NACK
SPI	Extra wire per slave	No	Full	None	8Mbps+	No	None
CAN	Large	Yes	Half	16-bit CRC ACK/NACK Differential signaling	1Mbps	No (for single frame)	Start, 11-bit ID, 6-bit control, 16-bit CRC, 2-bit ACK, 7-bit EOF

Figure 6: Karşılaştırma

3 ADC (Analog to Digital Converter)

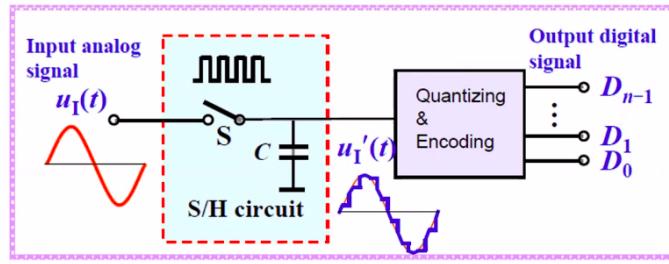


Figure 7: ADC

- 3 aşamadan oluşur:
 - Örnekleme (zamanda ayrık hale getirme) (İdeal örnekleme (Nyquist Teoremi) => Max. frekansın en az iki katı).
 - Kuantalama (Nicemleme) (genlikte ayrık hale getirme).
 - Kodlama (İkililik tabana çevirme).
- Kuantalama: sayısal sinyale dönüştürürken sonuca etki eden en küçük analog sinyalin değişimi (V : reference voltage range, N : number of bits in digital output, 2^N : number of states, ΔV : resolution):

$$\Delta V = \frac{V_r}{2^N}$$

Example 1: Temperature range of 0 K to 300 K to be linearly converted to a voltage signal of 0 to 2.5 V, then digitized with an 8-bit A/D converter.

$$2.5 / 2^8 = 0.0098 \text{ V, or about } 10 \text{ mV per step}$$

$$300 \text{ K} / 2^8 = 1.2 \text{ K per step}$$

$$(111)_2 \rightarrow 7 \times 5/8$$

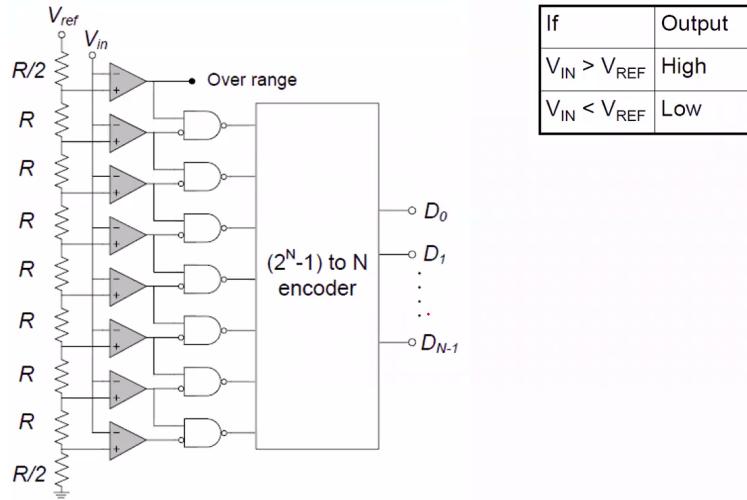
$$(000)_2 \rightarrow 0 \times 5/8 = 0$$

Example-2:

- Assumes the input analog voltage is changing between 0-5 V.
- Using a 3-bit A/D converter draw the output as the input signal ramps from 0 to 5V.
- Calculate the resolution in volts. $5/2^3\text{V}$
- What is the maximum possible voltage out? (this is called the full-scale output) (5 – Resolution)
- If the output is 011, what is the input? $3 \times 5/8 \text{ Volt}$

- Resolution'u artırmak kaliteyi arttırmır. Sampling rate'i artırmak kaliteyi arttırmır.

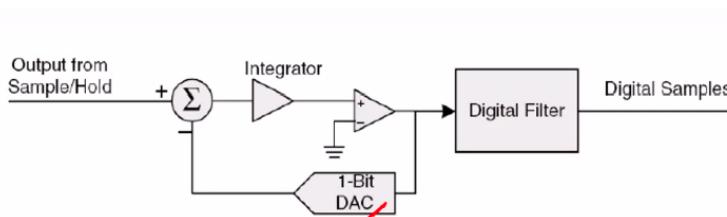
3.1 Flash ADC



8 bitlik bir adc. $V_{ref} = 8V$. Total $R = 8R = 7.5R + 0.5R$. En uçtaki değer $0.5V$, bir üstü $1.5V$, $2.5, 3.5\dots$ $V_{in} = 1.6V$ olsun. $1.6 > 0.5 = 1$, $1.6 > 1.5 = 1$, kalanlar 0.

- Çok hızlı (en hızlı) ve basit.
- Pahalı, girişte aksaklılıklar üretmeye eğilimli.

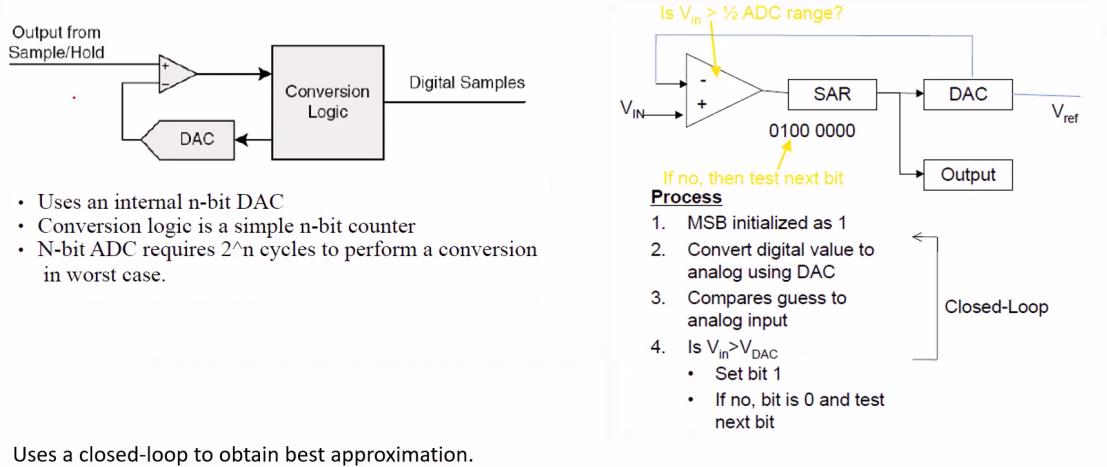
3.2 Sigma-Delta ADC



High sampling rate istiyor. Audio uygulamaları için birebir.

- input oversample'lanıp integrator'e gidiyor.
- integration daha sonra ground ile karşılaştırılır.
- iterasyonlarla serial bit stream'i elde edilir.
- output 1lerden oluşan bir serial bit stream olur
- yüksek resolution, precision component'e ihtiyaç yok.
- Samplingten dolayı yavaş, low bandwidth için iyi sadece.

3.3 Successive Approximation (SAR)

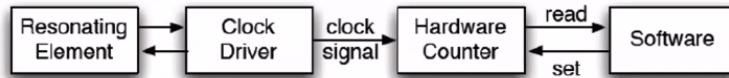


4 Genel Amaçlı Zamanlayıcılar ve Motor Kontrolü

Neden önemli:

- Scheduling a job in a system
- Signal sampling and generation
- Communication
- Navigation

4.1 Clock Sinyali Oluşturma



- Stabil (aynı frekansta) bir clock oluşturmak için bir resonating element gereklidir
 - Quartz (en yaygın)
 - MEMS
- Her zaman bir sapma payı vardır. Örneğin bir saat 2 yılda 21 dakika geri kahiyor olabilir. O zaman belli zaman aralıklarında saatı geri almak veya internete bağlanarak güncel saat bilgisini almak faydalı olabilir.
- Eğer dış dünyaya bir bağlantı yoksa sapma göz ardı edilebilir.
- Zamanlayıcılar interrupt üretebilir. (Zamanı geldiğinde belli bir işi çalıştırınmak gibi)

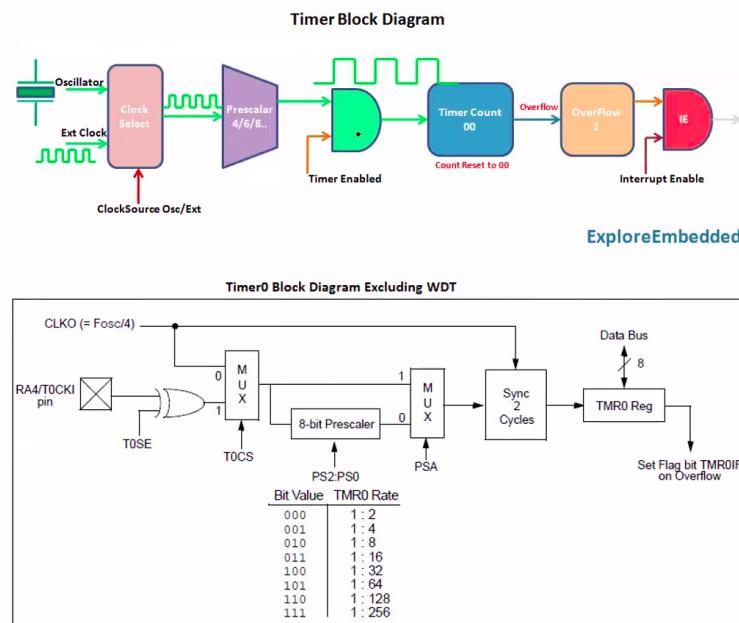
4.2 Timer'lar

- Mikrokontrolcüler içerisinde bir veya birden fazla timer bulundurabilir.
- Zamanlayıcılar, temel işlevlerini büyük ölçüde geliştirmek için genellikle konfigüre edilebilir.
- Frekansını, görev döngüsünü (duty cycle) veya her ikisini de ayarlayarak ihtiyaçlarımıza uygun olabilecek kare bir sinyal üretebilirler.
- General purpose timers, PWM, watchdog timers.

4.3 Bekçi Köpeği Zamanlayıcısı (Watchdog Timer)

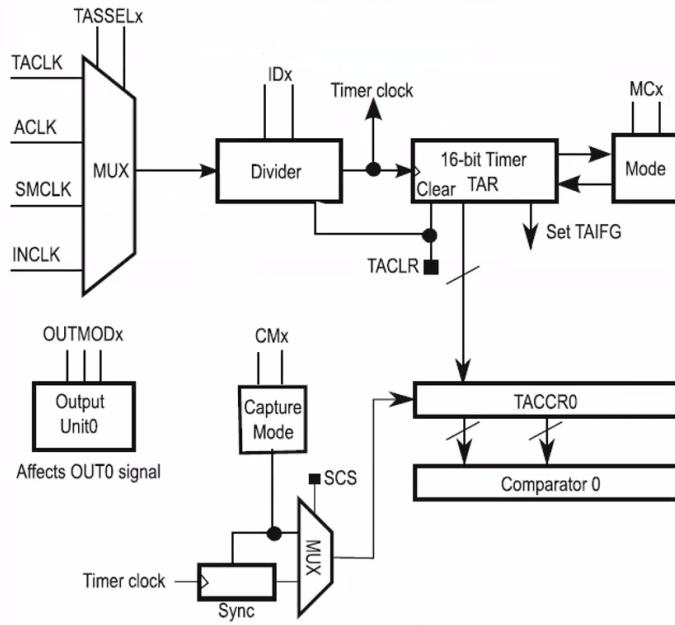
- Sistem herhangi bir yerde takıldığında (sonsuz döngü, null pointer, sonsuz veri bekleme vb.), bu durum tespit edilip sistemi resetleyebilen zamanlayıcılardır.
- Örnek:
 - Başta WDT, 5000 olarak set edilir ve her clockta bu değer azalır
 - WDT 0 olmasın diye kodun içinde belli yerlerde tekrar 5000 olarak set edilir
 - WDT, 0 oluyorsa demek ki kod ilerlemiyordur, bir yerde tıkanmıştır bu durumda timer resetlenecektir

4.4 PIC16F877 Timer Block Diagram



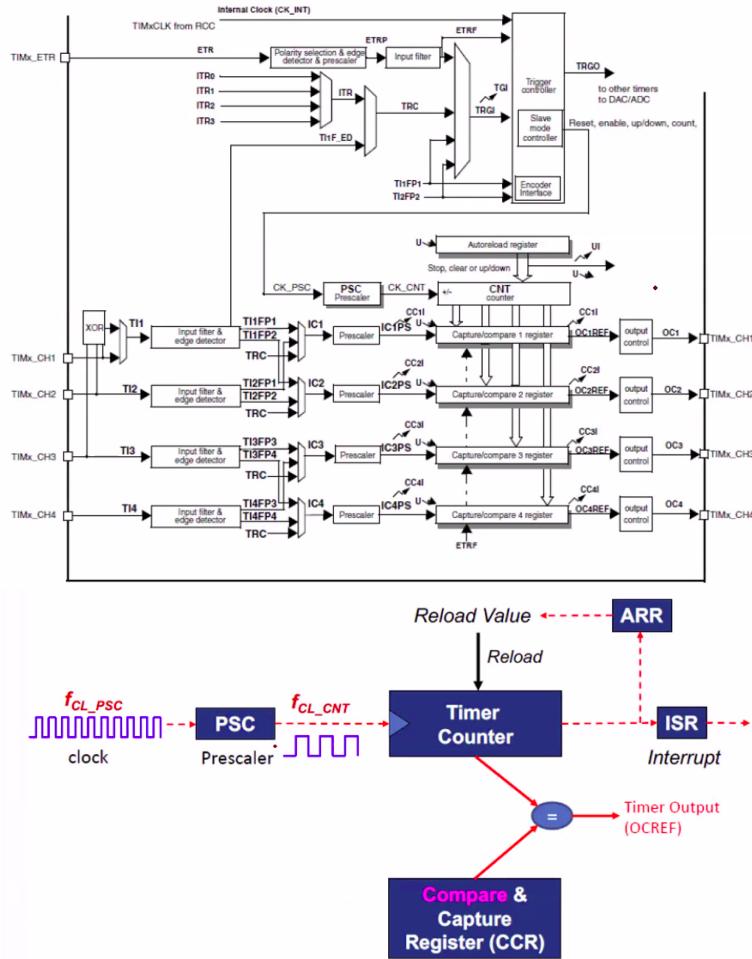
- T0SE: TMRO Source Edge Select Bit. 1: High to low, 0: Low to high
- T0CS: TMR0 Clock Source Select. 0: internal clock
- PSA: Prescaler assignment bit
- PS2:PS0: Prescaler rates.
- TMR0IF: Timer0 Interrupt Flag

4.5 MSP430 Timer Block Diagram

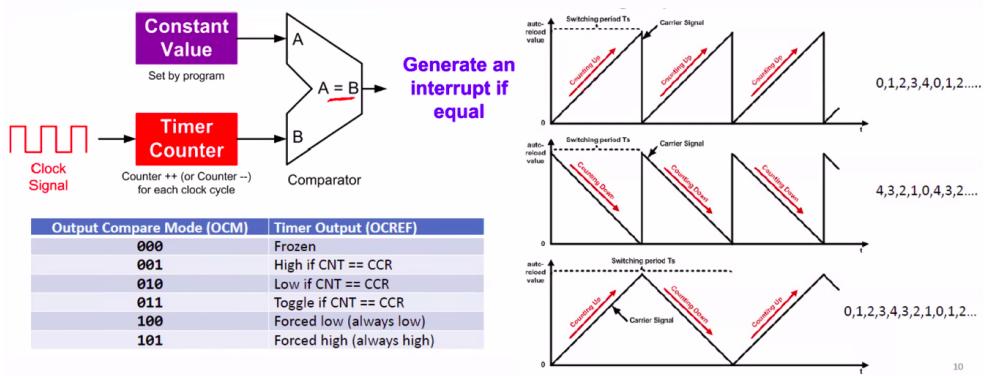


- Counts to 2^{16} .
- TASSELx: Selects clock source.
- IDx: Determines the frequency division.
- TACLR: Clears the counter.
- TAIFG: Timer a interrupt flag.
- TACCR0: Capture & Compare register.
- MCx: Counting up and down modes.

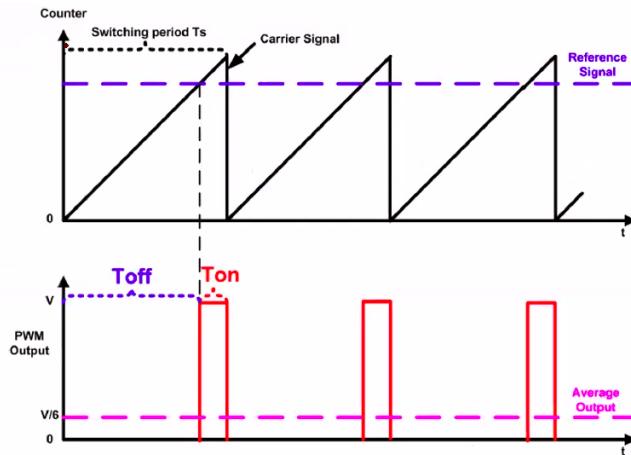
4.6 Timer in ARM Microcontrollers



- 4 farklı timer: 4 farklı interrupt.
- Autoreload register: sayma nereye kadar yapılacak.
- Compare interrupt üretebilir (eşitse mesela).
- Capture: timer değerini bir register'a atamak. (mesela bir butona art arda 2 kez bastık. İki basma arasındaki süreyi hesaplamak istiyoruz. Burada capture.)



4.7 PWM Modu



Mode	Counter < Reference	Counter \geq Reference
PWM mode 1 (Low True)	Active	Inactive
PWM mode 2 (High True)	Inactive	Active

- Doğrudan capture modu.
- Motor kontrolünde kullanılabilir.
- ARR: Üst sınır (autoreload register)
- CCR: Eşik değer (compare and capture register)
- Duty cycle (low true): $\frac{CCR}{ARR+1}$
- Duty cycle (high true): $1 - \frac{CCR}{ARR+1}$
- Period: $(1 + arr) \times \text{Clock period}$

4.8 Ultrasonik Mesafe Ölçümü

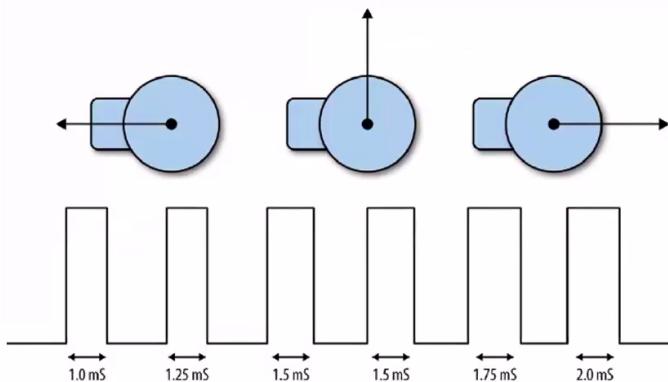
$$\text{distance} = \frac{(t_{\text{gidiş}} + t_{\text{dönüş}}) * v_{\text{ses}}}{2}$$

4.9 Motor Tipleri

	SERVO MOTOR	DC MOTOR	STEPPER MOTOR
Typical application	When high torque, accurate rotation is required.	When fast, continuous rotation is required.	When slow and accurate rotation is required.
Control hardware	Position is controlled through pulse width modulation (PWM). No controller required. May require PWM tuning.	Speed is often controlled through PWM. Additional circuitry required to manage power requirements.	Typically requires a controller to energize stepper coils. The RPi can perform this role, but an external controller is preferable and safer.

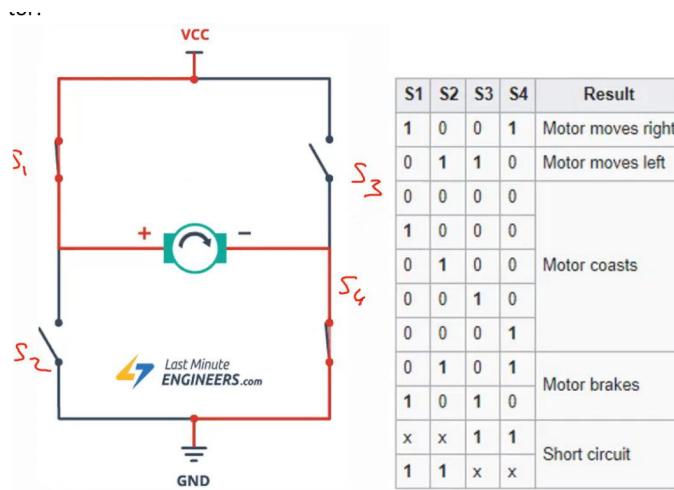
	SERVO MOTOR	DC MOTOR	STEPPER MOTOR
Control type	Closed-loop, using a built-in controller.	Typically closed-loop using feedback from optical encoders.	Typically open-loop, because movement is precise and steps can be counted.
Features	Known absolute position. Typically, limited angle of rotation.	Can drive very large loads. Often geared to provide very high torque.	Full torque at standstill. Can rotate a large load at very low speeds. Tendency to vibrate.
Example applications	Steering controllers, camera control, and small robotic arms.	Mobile robot movement, fans, water pumps, and electric cars.	CNC machines, 3D printers, scanners, linear actuators, and camera lenses.

4.9.1 Servo Motor



Servo motorun pozisyonu PWM darbesinin genişliği tarafından belirlenir.

4.9.2 DC Motor



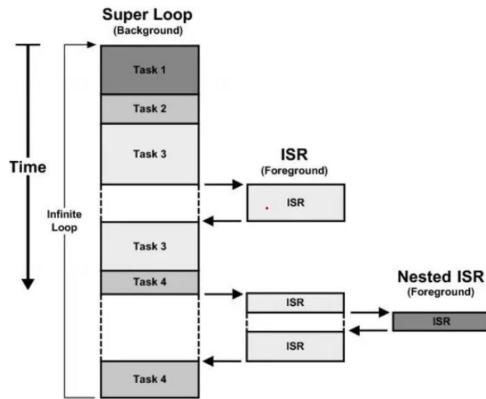
5 Gerçek Zamanlı Sistemler

Dışarıdan verilen bir girdiye, sonlu ve belirli periyotta cevap veren bilgi işleme sistemidir.

- Hard Real Time:
 - Deadline'ın aşılması çok büyük kayıplara sebep olur.
 - Örnek: Araç freni belli süreden fazla gecikirse ölüme sebep olur.
- Soft Real Time:
 - Deadline'ın aşılmasından sonra geçen süre arttıkça kayıp artar.
 - Örnek: Video yüklenirken ne kadar gecikirse QoS o kadar düşer.
- Real Time Sistemlerin özellikleri
 - Yanıt süresi deterministik ve sınırlıdır.
 - Eşzamanlılık sağlar.
 - Hatalara karşı toleranslıdır.
 - Yüksek çözünürlüklü timer bulunması önemlidir.
 - YANLIŞ: RT sistemler hızlı işlem yapar
 - YANLIŞ: Donanımlar güçlendirmek RT sistemlerin ihtiyacını karşılar
 - Düşük latency, yüksek throughput

5.1 Embedded Firmware

- I/O çevresel birimlerini kontrol eder.
- Fonksiyonel gereksinimlere göre istenilen çıktıları sağlamaya çalışır.
- Bare Metal (Super Loop) Yaklaşımı:



- Sonsuz döngüyle tasklar seri olarak execute edilir.
- Interrupt ve nested interrupt alabilir.
- Küçük ve yanıt süresinin çok önemli OLMADIĞI sistemler için uygun.
- Basit tasarım, az bellek kullanımı ve düşük cost.
- Herhangi bir hata olması durumunda tüm sistem etkilenir.

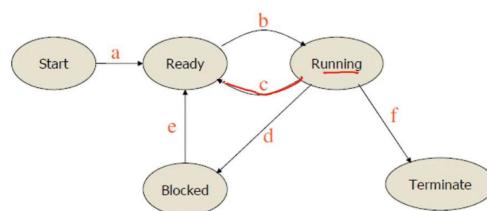
- **Embedded OS Tabanlı Yaklaşım:**

- RTOS, GPOS
- Scheduling, resource allocation ve multi tasking var.
- Arkaplan işlemleri de olabiliyor.
- Monolithic Kernel
 - * Tüm işletim sistemi elemanları tek bir programmış gibi çalışıyor.
 - * Linked together into a single large executable binary program
- Micro Kernel
 - * Kernel olabildiğince minimal tutulur.
 - * Her şey olabildiğince kernelin dışında, harici bir program olarak tutulur.
 - * Böylece bir hata olduğunda kernelin tamamı etkilenmez.
 - * Ayrıca sistem, böylece boş yere ihtiyacı olmayan bir sürü programı içermez.

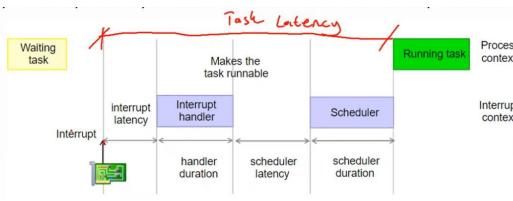
5.1.1 RTOS Temel Kavramlar

- Process kolsa, thread parmaklardır.
- Task ise en küçük bağımsız iş birimidir. (Thread'e denk düşer)
- RTOS'ta genelde 1 process bulunur ve her şey threadlerle yapılır.
- Thread Durumları:

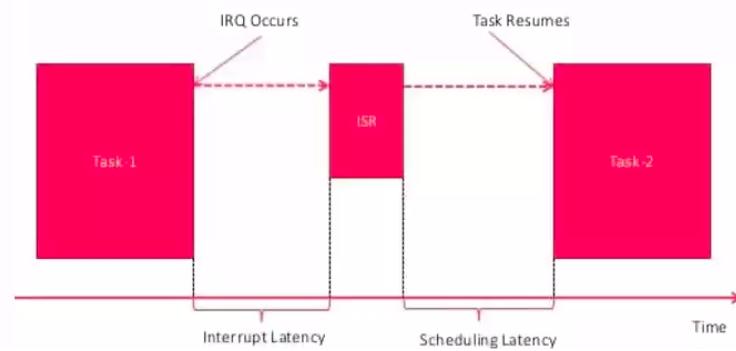
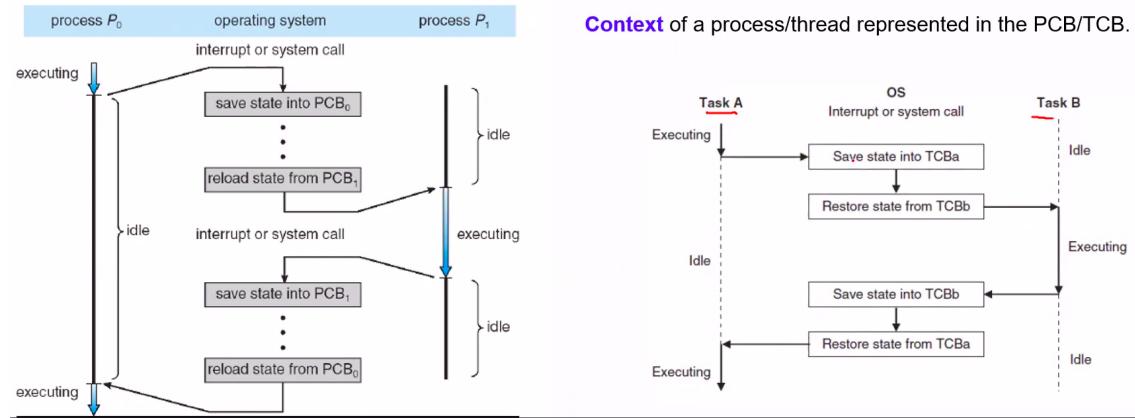
a)	Creation	The thread is created
b)	Dispatch	The thread is selected to execute
c)	Preemption	The thread leaves the processor
d)	Wait on condition	The thread is blocked on a condition
e)	Condition true	The thread is unblocked
f)	Exit	The thread terminates



- Preemption: Öncelikli taskın daha önce çalışması
- Context Switch: Tasklar arası geçiş için harcanan süre



- Interrupt Latency: Şu anki tasktan daha öncelikli bir interrupt geldiği anda şu anki taskın durdurulması ve öncelikli olan interrupt'a geçilmesi sırasında geçen süre.
- Handler Duration: Interrupt'ın çalışması + işlemler bittikten sonra maske kaldırılarak interrupt alabilir duruma geçme sırasında geçen süre
- Scheduler (Latency + Duration):
 - Başka interrupt gelebilir
 - System call çalışıyor olabilir.
 - Daha yüksek öncelikli tasklar varsa, onlar beklenenecek.
 - Context switch süresi.
- Dispatch latency sabit (task switching).



6 Scheduling

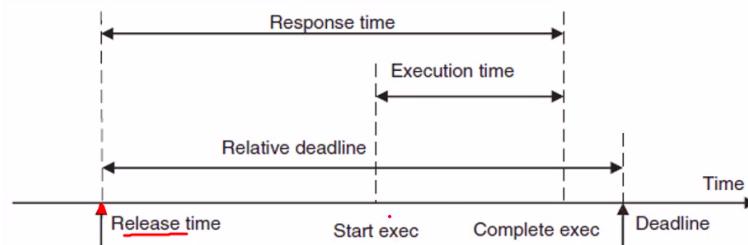
- Preemptive vs Non-Preemptive Scheduling:

- Non-preemptive (bölnemeyen) scheduling'de bir task bitmeden başka task başlayamaz.
- Preemptive scheduling'de öncelikli bir task geldiği zaman diğer task bölünüp öncelikli olan task çalışır, bittiğinde ise bölünen taska devam edilir.

- Task Tipleri:

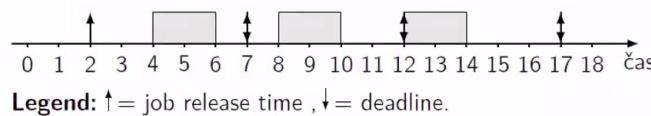
- Periodic Task: Belli periyotta tekrar eden tasklar
- Aperiodic Task: Tek seferlik event-driven tasklar
- Sporadic Tasks: Belirli bir zaman aralığında gerçekleşen (event-driven şekilde başlatılan) periyodik tasklar

6.1 Task Specification



- τ_i : Task: i numaralı task
- T_i : Period: Periyodik taskın periyodu
- r_i : Release time: Task'ın geldiği an
- d_i : Deadline: Task'ın biteceği an
- D_i : Relative deadline: Release time'dan deadline'a kadar olan süre
- C_i : Execution time: Task'ın çalışma süresi
- R_i : Response time: Task'ın geldiği andan bittiği ana kadar geçen süre
- (T_i, C_i, D_i)

Periodic task τ_i with $r_i = 2$, $T_i = 5$, $C_i = 2$, $D_i = 5$ can be executed like this (continues until infinity).



- Periyotla İlgili Bazı Kavramlar:

- Hyperperiod: $\{T_1, T_2, \dots, T_n\}$ şeklinde verilen periyotların EKOK'u.
- Task Utilization: $u_i = \frac{C_i}{T_i}$
- System Utilization: $U = \sum_{i=1}^N u_i$

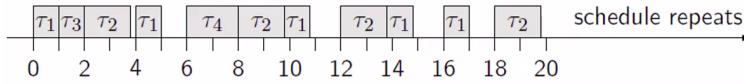
6.2 Yaygın Scheduling Yaklaşımları

- Off-Line Scheduling (Static, Clock-Driven):

- Her taskin release time ve execution time bilgilerine göre önceden hesaplamalar yapılarak bir tablo (hangi taskin hangi zamanda yürütüleceğinin yazıldığı) oluşturulur ve bu tabloya göre scheduling yapılır.
- Çalışma sırasında anlık olarak schedule değişmez.
- Scheduling hatası olmayacağı garanti edilebilir, çünkü net ve basit bir yapı kullanılmıştır.
- Flexible değil, sürekli olarak task ekleyip çıkarmaya müsait değil çünkü tek bir değişiklikte tüm tablo tekrar hesaplanır.
- Çok fazla task olursa tablo çok büyür, gömülü sistemlerin hafızası oldukça sınırlıdır.

System of four tasks:

	τ_1	τ_2	τ_3	τ_4
Period	4	5	20	20
Execution time	1	1.8	1	2



The schedule table will look like this:

Time	0	1	2	3.8	4	5	6	8	9.8	10.8	...	18
Task	τ_1	τ_3	τ_2	X	τ_1	X	τ_4	τ_1	X	τ_2	...	τ_2

- Frame-Based Yaklaşım:
 - * Her bir frame için preemption yok
 - * Her iş frame'e sığmalı
 - * Schedule tablosunun hesaplanması dışında, olası hata durumları da hesaplanmalı (task overrun gibi)
 - * Frame size yani f değeri seçilmeli:
 - $f \geq \max_{i=1,\dots,n} C_i$
 - Tabloyu küçük tutmak için f , hiperperiyodu tam bölebilmeli.
 - $2 \times f - \text{EBOB}(T_i, f) \leq D_i$ olmalı.

■ Let's have a system from the last example with four tasks:
 $\tau_1 = (4, 1)$, $\tau_2 = (5, 1.8)$, $\tau_3 = (20, 1)$, $\tau_4 = (20, 2)$.

1) If $f = 2$,

$$2 \times 2 - \text{GCD}(4, 2) \leq 4 \rightarrow 4 - 2 \leq 4$$

■ From the first constraint (1): $f \geq 2$.

$$2 \times 2 - \text{GCD}(5, 2) \leq 1.8 \rightarrow 4 - 1 \leq 5$$

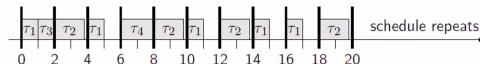
■ Hyperperiod is 20 so second constraint (2), tells us that f can be one of 2, 4, 5, 10 a 20.

$$2 \times 2 - \text{GCD}(20, 2) \leq 20 \rightarrow 4 - 2 \leq 20$$

■ Third constraint (3) satisfies only $f = 2$.

$$2 \times 2 - \text{GCD}(20, 2) \leq 20 \rightarrow 4 - 2 \leq 20$$

■ Possible cyclic schedule:

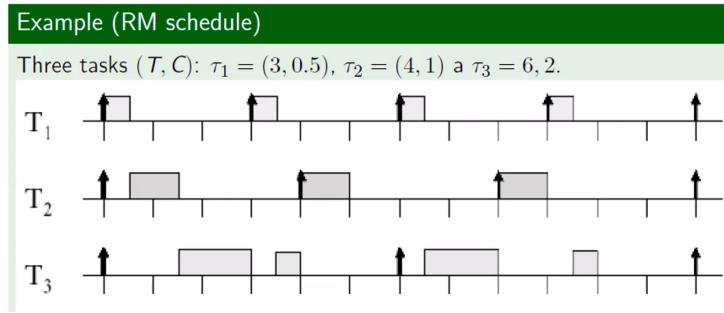


- On-Line Scheduling (Dynamic, Runtime):

- Static-Priority Scheduling (VxWorks, SCED_FIFO):

- * Round Robin:
 - Aynı öncelike sahip tasklar, eşit zaman aralıklarına bölünerek çalıştırılır
 - Yüksek öncelikli bir task varsa, kesinlikle bu task çalışır, bu bitmeden düşük öncelikli tasklar çalışmaz.

- * Rate-Monotonic:

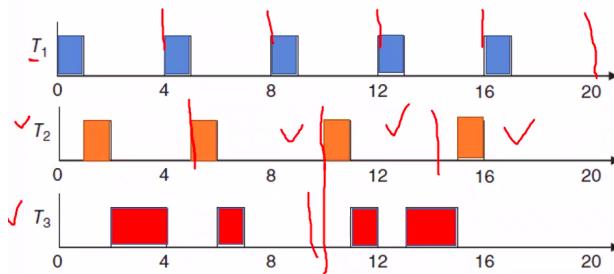


Example 4.9 Task missing deadline by RM

Figure 4.13 shows that when we schedule three periodic tasks

$$\underline{\tau_1} = (4, 1), \underline{\tau_2} = (5, 2), \underline{\tau_3} = (10, 3.1). \rightarrow (\text{Period}, \text{Execution Time})$$

according to the RM algorithm, the first instance of T_3 misses its deadline – it has remaining 0.1 units of execution time that is not completed by its deadline 10.



- En küçük periyotlu taskin önceliği daha yüksektir, önce o çalıştırılır.
- Periyodu yüksek olana sıra gelmeyebilir veya düşük periyotlu yeni işler geldiğinde preemption olduğundan diğer işler sürekli bölünüp deadline kaçırılabilir.

- * Deadline-Monotonic:

Deadline-Monotonic priority assignment

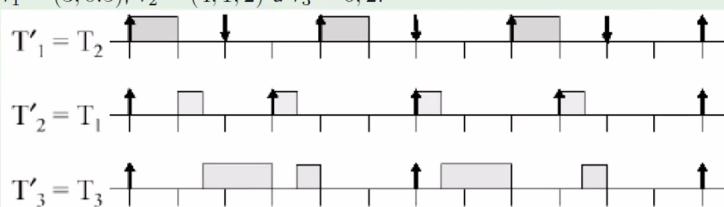
The earlier deadline D_i , the higher priority p_i .

Pro each two tasks τ_i a τ_j : $D_i < D_j \Rightarrow p_i > p_j$.

Example (DM schedule)

Let's change the RM example by tightening deadline of $\tau_2 = (T, C, D)$:

$\tau_1 = (3, 0.5)$, $\tau_2 = (4, 1, 2)$ a $\tau_3 = 6, 2$.



- En yakın deadline olan task, en önceliklidir.

- Deadline-Driven Scheduling
- General Purpose OS Scheduling
- **Dynamic Priority Scheduling**

- Earliest Deadline First

- * Optimal if single processor and preemptive.

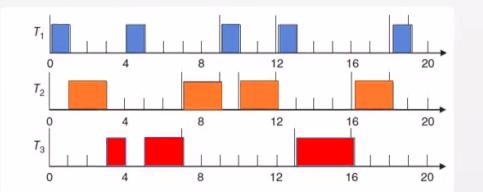
$$T_1 = (4, 1), T_2 = (5, 2), T_3 = (10, 3.1)$$

three independent, preemptable periodic tasks

At 8, the third instance in T_1 arrives. Its deadline is at 12, later than the deadline of the T_2 instance in execution, which still has 1.1 units to complete. So, T_2 continues to run and completes at 9.1.

At 9.1, T_1 is the only task waiting to run, and so it executes.

At 10, an instance in T_2 is released with deadline 15, and meanwhile, an instance in T_3 is released with deadline at 20. So, the T_1 instance continues to run. Both T_2 and T_3 are waiting.



- Least Slack Time

- * $t_{REM} = C_i - (t - R_i)$
- * $d_i - t - t_{REM}$
- * En düşük slack time en yüksek öncelik

Least Slack Time (LST)

• Remaining execution time : $t_{REM} = C_i - (t - R_i)$
• Slack time (laxity): $l = t_{slack} = d_i - t - t_{REM}$

	R	C	d
T1	0	10	33
T2	4	3	28
T3	5	10	29

Requires calling the scheduler periodically, and to recompute the laxity.

When $t=4$,

$$l(T1) = d_1 - t - t_{REM} = 23$$

Diagram showing the Gantt chart for LST scheduling:

- Task T1 starts at t=0 and ends at t=10. $t_{REM} = 6$ ms.
- Task T2 starts at t=4 and ends at t=7. $t_{REM} = 3$ ms.
- Task T3 starts at t=5 and ends at t=15. $t_{REM} = 10$ ms.
- Slack times at t=4: $l(T1)=33-5-6=22$, $l(T2)=28-5-2=21$, $l(T3)=29-5-10=14$.
- Slack times at t=10: $l(T1)=33-17-6=10$, $l(T2)=28-17-2=9$.