Embedded Systems Summary

Nicolas Trüssel

August 4, 2016

1 Miscellaneous

Dependable means

- Reliable (continue working correctly when it worked correctly before)
- Maintainable (recover from errors)
- Available (probability that it's working)
- Safety
- Security

Efficient in terms of:

- Energy
- Code size
- Memory consumption
- Run time
- Weight
- Cost

2 Generic Time Triggered Cyclic Executive Scheduler

Let f denote the frame length, P the full period, $D(k)$ the relative deadline of task k, $C(k)$ its execution time, and $p(k)$ the period of task k (how often it occurs). Then the following conditions have to be satisfied:

- $\forall k.f \leq p(k)$ (at most one execution within a frame)
- $P = \text{lcm}_k\,(p(k))$
- $\forall k.f \geq C(k)$ (processes start and complete within single frame)
- $\forall k.2f - \gcd\,(p(k), f) \leq D(k)$ (between release time and deadline of every task there is at least one frame boundary)

3 Aperiodic Scheduling

	Equal arrival, non-preemptive	Arbitrary arrival, preemptive
Independent Tasks	EDD	EDF
Dependent Tasks	LDF	EDF*

3.1 Definitions

- $J = \{j_1, j_2, \ldots, j_n\}$ is a set of tasks.
- a_i or r_i is the arrival / release time of task i.
- d_i is the deadline of task i.
- C_i is the total computation time of task i.
- $c_i(t)$ is the the remaining execution time of task i at time t.
- s_i is the start time of task i.
- f_i is the finish time of task i.
- $L_i = f_i - d_i$ is the lateness of task i.
- $E_i = \max(0, L_i)$ is the exceeding time or tardyness of task i.
- $X_i = d_i - a_i - C_i$ is the laxity or slack of task i.
- $w_i = s_i - a_i$ is the waiting time of task i.
- $R_i = f_i - a_i$ is the response time of task i.

3.2 EDD

Schedule the tasks in order of non-decreasing deadlines. This minimizes the maximal lateness.

3.3 EDF

Always execute the task with the earliest absolute deadline. Schedulability test:

$$\forall i \in [n].t + \sum_{k=1}^{i} c_k(t) \leq d_i$$

3.4 LDF

Among all tasks without successors select the task with the latest deadline. Put it in a stack. Repeat until no more tasks. Now execute tasks as they are on the stack.

3.5 EDF*

Modify arrival and deadline of each task and use EDF on modified tasks.

$$r_j^* = \max_j \left(r_j, \max_i \left(r_i^* + C_i | J_i \to J_j \right) \right)$$

$$d_i^* = \min_i \left(d_i, \min_j \left(d_j^* - C_j | J_i \to J_j \right) \right)$$

4 Periodic Scheduling

Periodic scheduling is always preemptive.

	Deadline = Period	Deadline ≤ Period
Static Priority	Rate Monotonic	Deadline Monotonic
Dynamic Priority	EDF	EDF*

4.1 Definitions

- $\Gamma = \{\tau_1, \tau_2, \dots \tau_n\}$ is set of periodic tasks.
- $\tau_{i,j}$ is the j-th instance of task τ_i.
- $a_{i,j}, r_{i,j}, d_{i,j}, s_{i,j}, f_{i,j}$ are the same as for aperiodic tasks.
- Φ_i is the phase of task i (release time of the first instance).
- D_i is the relative deadline of task i.
- T_i is the period of the task (time between 2 releases).

The following hypotheses are assumed

- $r_{i,j} = \Phi_i + (j-1)T_i$
- C_i is constant.
- $d_{i,j} = \Phi_i + (j-1)T_i + D_i$
- The tasks are independent

4.2 Rate Monotonic

Always schedule the task that has the shortest period. Sufficient (but not necessary) schedulability test:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$$

4.3 Deadline Monotonic

Always schedule the task that has the shortest relative deadline. Sufficient (but not necessary) schedulability test:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$$

4.4 Necessary and sufficient schedulability test

We define the interference I_i for task i as

$$I_i(t) = \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

where the tasks are ordered such that $m < n \iff D_m < D_n$

4.4.1 Algorithm

The tasks are ordered as described above.

```
foreach (τ_i ∈ Γ) {
        I = 0;
        do {
                R = I + C_i;
                if (R > D_i) return false; // unschedulable
                I = Σ_{j=1}^{i-1} ⌈R/T_j⌉ C_j;
        } while (I + C_i > R);
}
return true;
```

The final values `R` are the longest response times.

4.5 EDF

Always schedule the task with the earliest deadline. For $D_i = T_i$ the tasks are schedulable with EDF iff $\sum_{i=1}^{n} \frac{C_i}{T_i} = U \leq 1$.

If $D_i \leq T_i$, then the test $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$ is sufficient but not neccesary.

5 Mixed Task Sets

A simple solution would be to schedule asynchronous tasks/events whenever there is time available. However this might lead to starvation.

5.1 Polling Server (RM)

A polling server is an additional periodic task which is used to serve aperiodic requests. An aperiodic task is schedulable if (sufficient but not necessary):

$$\left(1 + \left\lceil \frac{C_a}{C_s} \right\rceil\right) T_s \leq D_a$$

Where C_a, C_s, D_a, T_s are the total computation time of the aperiodic request, the computation time of one server task, the deadline of the aperiodic request, and the period of the server task. It is assumed that one aperiodic request is served before another task arrives.
When no task is available and the server has highest priority, the server looses its timeslot to the other tasks.

5.2 Total Bandwidth Server (EDF)

A certain cpu utilization (bandwidth) is reserved for the server. When the kth aperiodic request arrives it receives the deadline

$$d_k = \max\left(r_k, d_{k-1}\right) + \frac{C_k}{U_s}$$

where C_k is the total execution time of the request and U_s is the utilization factor of the server. The request is then treated as a normal periodic task.
The schedule is feasible iff $U_p + U_s \leq 1$. Where U_p is the processor utilitation of the periodic tasks.

6 Priority Inheritance Protocol

- Tasks have both nominal priority P_i and active priority $p_i \geq P_i$. The tasks are sorted by nominal priority, highest first.
- Tasks are scheduled by their active priority using FCFS for ties.
- When J_i is blocked, it transmits its active priority p_i to the blocking task, $p_k = p_i$
- When J_k exits the critical section, the highest priority job blocked by J_k is awakened. If no other jobs are blocked by J_k, p_k is set to P_k otherwise to the highest priority of the jobs blocked by J_k.
- Priority inheritance is transitive.

Tasks have fixed priority (i.e. the kernel is not allwed to decrease the priority of a task below p_i).

7 Communication

7.1 Random Access

Best sending rate is $p = \frac{1}{n}$ (using maximization of $p\,(1-p)^{n-1}$) leads to utilization of $\frac{1}{e}$.

7.2 TDMA

Statically allocated slots, synchronization done by master.

7.3 CSMA/CD

Detect collitions, do exponential backoff of $\{0, \dots 2^n - 1\}$ for nth collision (possibly with upper bound on n). The wait time is twice the round trip time.

7.4 Token Ring

Pass token around, attach to token.

7.5 CSMA/CA

IMAGE HERE

7.6 CSMA/CR

Before message is sent, arbitration happens. During arbitration every client can detect which one is allowed to send. During arbitration, every node sends its unique id and checks whether the result on the bus matches the id. Low bits override high bits, hence the one with the lowest id wins.

7.7 FlexRay

FlexRay uses a combination of TDMA and something close to CSMA/CA alternating, first a static segment using TDMA, then a dynamic segment

7.8 Bluetooth

Frequency hopping in range $2402 + k$ MHz, $0 \leq k \leq 78$, 1600hops per second. Organized in piconets with 1 master and at most 7 slaves, which can be combined into scatternets. A node is a master in at most one piconet and can be a slave in multiple piconets.
The master can only send in even time slots. After each slot, the frequency is changed using a pseudo-random sequence based on the master's device address (unique) and the phase of the master's system clock. Packets are either 1, 3, or 5 slots long, the frequency is only changed after the full packet is sent, however it changes 1, 3, or 5 steps, depending on the packet length.
The initial setup between master and slave is done by the inquiry step, where the master sends it s ID using a special channel sequence, the slave listens and replies with its ID, clock and other information.
Then the connection is started with in the page mode. The master sends its and the slaves address using a special channel sequence, the slave listens whether it hears its address and answers with its own address. The master then sends a frequency hop synchronization packet (FHS) to the slave which allows them to synchronize and connnect.
A slave can be in multiple states, active, hold (not processing date packtes), sniff (awaken in regular time intervals), and park (no connection but still synchronized).

7.9 Packet Details

Total packet length is either 366 bits, 1626 bits or 2870 bits for 1, 3, and 5 slot packets respectively. Packet length in bits corresponds to μs send time. For unprotected data, the payload is 216 bits, 1464 bits and 2712 bits.

8 Low Power Design

- α switching activity
- C_L load capacity
- f clock frequency
- V_{dd} supply voltage
- V_T threshold voltage ($V_T \ll V_{dd}$)

$$
\begin{aligned}
P &\sim \alpha C_L V_{dd}^2 f \\
\tau &\sim C_L \frac{V_{dd}}{(V_{dd} - V_T)^2} \\
f_{max} &\sim V_{dd} \\
E &\sim \alpha C_L V_{dd}^2 \underbrace{(\#\text{cycles})}_{f \cdot t}
\end{aligned}
$$

8.1 YDS Algorithm

The YDS algorithm can be use to schadule tasks with dynamic voltage scaling such that the engergy consuption is minimal.
We define

$$
\begin{aligned}
V'([z, z']) &= \{v_i \in V : z \leq a_i < d_i \leq z'\} \\
G([z, z']) &= \sum_{v_i in V'([z,z'])} \frac{C_i}{z' - z}
\end{aligned}
$$

Algorithm:

1. Calculate intensities
2. Schedule jobs in highest intensity interval using EDF, intensity as frequency
3. Adjust arrival times and deadlines by excluding the interval, restart

The online version continuously updates best schedule for all available tasks. It has at most 27 times higher energy consumption.

8.2 DPM

With dpm, make sleep periods as long as possible, reduce the amount of on/off transitions. Use YDS and round frequencies up.

9 Models

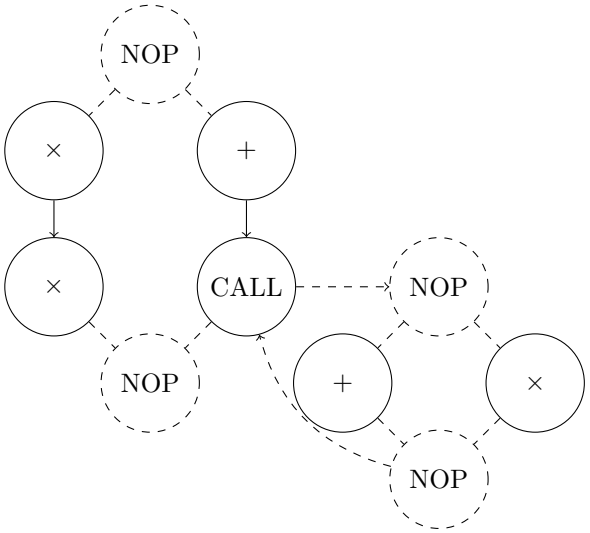
9.1 Sequence Graph

A sequence graph is a hierarchy of directed acyclic dependence graphs with dedicated start and end nodes. There are two kinds of nodes: task nodes and hierarchy nodes (CALL for module calls, BR for branches, LOOP for iterations).

9.1.1 Examples

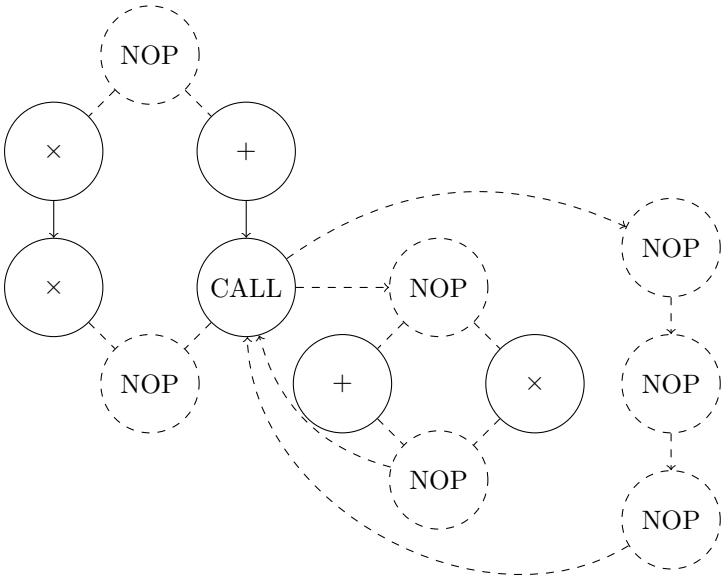
```
x = a * b;
y = x * c;
z = a + b;
submodule(a, z);

void submodule(int m, int n) {
        p = m + n;
        q = m * n;
}
```




```
x = a * b;
y = x * c;
z = a + b;

if (z > 0) {
        p = m + n;
        q = m * n;
}
```


NOP
×
+
×
CALL
NOP
NOP
NOP
+
×
NOP
NOP
NOP

10 Synthesis

10.1 Resource Graph

A resource graph is a graph $G_R = (V_R, E_R), V_R = V_S \cup V_T$, where V_S are the nodes from the sequence graph and V_T are the different resource types. It is bipartite, $E_R = V_S \times V_T, (v_s, v_t) \in E_R$ means v_s can be executed on resource type v_t. Additionally there is a cost function $c : V_T \to \mathbb{Z}$ and a execution time function $w : E_R \to \mathbb{Z}^{\geq 0}$
Furthermore $\alpha(v_t)$ denotes the number of available instances of resource type v_t, $\beta(v_s)$ denotes on which resource type v_s is running, and $\gamma(v_s)$ denotes on which instance of this resource type it is running.

10.2 Scheduling

A schedule is a function $\tau : V_S \to \mathbb{Z}^{>0}$ that determines the starting times of operations. It is feasible if

$$\forall (v_i, v_j) \in E_S . \tau(v_j) - \tau(v_i) \geq w(v_i)$$

where $w(v_i) = w(v_i, \beta(v_i))$ denotes the execution time of v_i.
The latency L is $L = \tau(v_n) - \tau(v_0)$, the differnce between the starting times.

10.2.1 ASAP

```
ASAP(V_S, E_S, w) {
        \tau(v_0) = 1
        do {
                v_i ← {v ∈ V_S|v's predecessors are planned};
                τ(v_i) = max {τ(v_j) + w(v_j)|(v_j, v_i) ∈ E_S};
        } while (unplanned operations exist);
}
```

10.3 ALAP

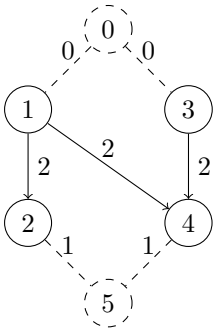
```
ALAP(V_S, E_S, w, L_max) {
        τ(v_n) = L_max + 1;
        do {
                v_i ← {v ∈ V_S|v's successors are planned};
                τ(v_i) = min {τ(v_j)|(v_i, v_j) ∈ E_S} − w(v_i);
        } while (unplanned operations exist);
}
```

10.4 Bellman Ford

One might add additional constraints as weighted edges in the sequence graph, which can be resolved using Bellman Ford to find the single source longest path. Set the weight of the normal edges to their execution times.

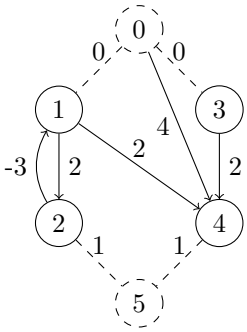
```
BELLMAN-FORD(V_S, E_S, W) {
        τ_0^0 = 0;
        for (int i = 0; i < |V_S|; i++) τ_i^0 = w(v_0, v_i);
        for (int j = 0; i < |V_S|; i++) {
                for (int i = 0; i < |V_S|; i++) {
                        τ_i^{j+1} = max {τ_i^j, τ_k^j + w(v_k, v_i)|k ≠ i};
                }
                if (∀i . τ_i^{j+1} = τ_i^j) return true; // success
        }
        return false; // failed
}
```

10.4.1 Example constraints



We add the following constraints

- Between 1 and 2 there are at most 3 time units
- Between 0 and 4 there are at least 4 time units



10.5 List Scheduling

```
LIST(V_S, E_S, V_R, E_R, α, β, priorities) {
        V_T = V_R − V_S;
        t = 1;
        do {
                foreach(v_k ∈ V_T) {
                        U_k = candidates to be scheduled;
                        T_k = running operations;
                        S_k = subset of U_k with maximal priority and |S_k| + |T_k| ≤ α(v_k);
                        foreach(v_i ∈ S_k) { τ(v_i) = t; }
                }
                t = t + 1;
        } while (v_n unplanned);
}
```

10.6 Integer Linear Programming

To get optimal results one can use ILP. First, for each $v_i \in V_S$ we have to determine l_i and h_i, the earliest and latest starting time respectively, using ASAP and ALAP with a suitable L_{max}. $x_{i,t} = 1 \iff$

operation v_i starts at time t.

$$\begin{gathered}
\min \tau(v_n) - \tau(v_0) \qquad \text{subject to} \\
\forall v_i \in V_S \forall l_i \leq t \leq h_i \,.\, x_{i,t} \in \{0,1\} \\
\forall v_i \in V_S \,.\, \sum_{t=l_i}^{h_i} x_{i,t} = 1 \\
\forall v_i \in V_S \,.\, \sum_{t=l_i}^{h_i} t \cdot x_{i,t} = \tau(v_i) \\
\forall (v_i, v_j) \in E_S \,.\, \tau(v_j) - \tau(v_i) \geq w(v_i) \\
\forall v_k \in V_T \forall 1 \leq t \leq \max_i \{h_i\} \,.\, \sum_{\forall i.(v_i,v_k) \in E_R} \sum_{p'=\max(0,t-h_i)}^{\min(w(v_i)-1,t-l_i)} x_{i,t-p'} \leq \alpha(v_k)
\end{gathered}$$

10.6.1 Modifications

To adapt the ILP to iterative algorithms (marked graphs, pipelining), replace

$$\begin{gathered}
\forall (v_i, v_j) \in E_S \,.\, \tau(v_j) - \tau(v_i) \geq w(v_i) \\
\forall v_k \in V_T \forall 1 \leq t \leq \max_i \{h_i\} \,.\, \sum_{\forall i.(v_i,v_k) \in E_R} \sum_{p'=\max(0,t-h_i)}^{\min(w(v_i)-1,t-l_i)} x_{i,t-p'} \leq \alpha(v_k)
\end{gathered}$$

by

$$\begin{gathered}
\forall (v_i, v_j) \in E_S \,.\, \tau(v_j) - \tau(v_i) \geq w(v_i) - d_{i,j} \cdot P \\
\forall v_k \in V_T \forall 1 \leq t \leq \max_i \{h_i\} \,.\, \sum_{\forall i.(v_i,v_k) \in E_R} \sum_{p'=0}^{w(v_i)-1} \sum_{\forall p \,.\, l_i \leq t-p'+p \cdot P \leq h_i} x_{i,t-p'+p \cdot P} \leq \alpha(v_k)
\end{gathered}$$

and
where $d_{i,j}$ is the amount of tokens on edge (i, j).

10.7 DVS ILP

$$\begin{gathered}
\min \sum_{k \in K} \sum_{v_i \in V_S} y_{ik} \cdot e_k(v_i) \qquad \text{subject to} \\
\forall v_i \in V_S, k \in K \,.\, y_{ik} \in \{0,1\} \\
\forall v_i \in V_S \,.\, \sum_{k \in K} y_{ik} = 1 \\
\forall (v_i, v_j) \in E_S \,.\, \tau(v_j) - \tau(v_i) \geq \sum_{k \in K} y_{ik} \cdot w_k(v_i) \\
\forall v_i \in V_S \,.\, \tau(v_i) + \sum_{k \in K} y_{ik} \cdot w_k(v_i) \leq d(v_i)
\end{gathered}$$

where K is the set of voltage levels and there are no resource constraints.