Níže je **kompletní, přesná a plně detailní rekapitulace** všech rozhodnutí, pravidel, architektonických závěrů a technických principů, na kterých jsme se společně shodli pro návrh funkce **LCNC – STARTING**. Je to dlouhé, strukturované a přesné, jak jste požadoval. Slouží jako uzavření první etapy návrhu architektury.

==========

**REKAPITULACE CELÉ ARCHITEKTURY

LINUXCNC – STARTING STATE (OFF → STARTING → ON)**

==========

Rozděleno do sekcí:

1. **Motivace a cíle**
2. **Nový stavový automat (OFF → STARTING → ON)**
3. **Konfigurace v INI**
4. **Nové HAL piny – domény a názvosloví**
5. **Sekvenční logika STARTING**
6. **Čtení a správa READY signálů**
7. **Timeout STARTING + logika selhání**
8. **Interrupce uživatelem (OFF během STARTING)**
9. **E-STOP a jiné prioritní stavy**
10. **Chybová hlášení GUI vs. terminál**
11. **Umístění logiky – TASK jako vlastník**
12. **Varianty, které jsme zamítli a proč**
13. **Další rozšiřitelné prvky (machine.state)**
14. **Shrnutí architektonických zásad**
15. **Co zbývá navrhnout v etapě 2 (implementační)**

==========

1. Motivace a cíle

==========

Pro výrobní stroje existují případy, kdy po zapnutí stroje (STATE_ON) nestačí jen povolit drivery os a vřetena. Profesionální drivery často poskytují signál **READY** a některé další subsystémy (hydraulika, pneumatika, PLC logika…) se musí připravit dříve, než lze stroj označit jako ON.

Cíle:

- Zavést nový mezistav **STARTING** mezi OFF a ON.
- Zajistit volitelnost této logiky v INI souboru.
- Nepoškodit existující instalace (výchozí = staré chování).
- Nepřidat povinnost modifikovat ostatní GUI.
- Zachovat architektonickou čistotu backendu LCNC.

==========

2. Nový stavový automat: OFF → STARTING → ON

==========

Nový stav:

```
EMC_TASK_STATE_STARTING
```

Startovací sekvence:

1. Task obdrží příkaz ON (NML state = EMC_STATE_ON).
2. Pokud INI říká STARTING=TRUE → stroj nepřejde rovnou do ON.

3. Task přepne do stavu STARTING.
4. Spustí sekvenční ověřování readiness.
5. Po úspěchu task nastaví být ve stavu ON.

V případě selhání STARTING → OFF.

==========

3. Konfigurace v INI

==========

Sekce:

```
[TASK]
STARTING = TRUE|FALSE
STARTING_TIMEOUT = <integer or -1>
```

Význam:

- `STARTING = TRUE` → aktivuje celý mechanismus.
- `STARTING = FALSE` → LCNC funguje jako dosud (OFF → ON).
- `STARTING_TIMEOUT = -1` → nikdy timeout.
- `STARTING_TIMEOUT > 0` → maximální délka sekvence v sekundách.
- `STARTING_TIMEOUT = 0` → NEPLATNÁ hodnota → způsobí chybu při načítání INI.

Volba sekce `[TASK]` je správná, protože STARTING logika je plně úkolem úrovně task (NML, state machine).

==========

4. Nové HAL piny – názvosloví a domény

==========

Shodli jsme se na přesných názvech HAL pinů.

A) Globální piny (machine-level)

Vytváří je **task**, protože jde o stav stroje jako celku.

TASK → HAL (stav STARTING)

```
machine.starting    (bit OUT)
```

HAL → TASK (globální readiness)

```
machine.ready       (bit IN)
```

Význam:

- `machine.starting` = task oznamuje, že běží startovací sekvence.
- `machine.ready` = integrátor oznamuje, že globální subsystémy jsou připraveny.

Použití domény `machine.*` je správné:

- nepatří to halui (UI modul)
- nepatří to motion (RT řízení pohybu)
- není to iocontrol (toolchanger/IO logika)
- je to přesně doména „stav stroje“

Prefix `task.machine.*` jsme zvažovali, ale zavrhli z ergonomických a architektonických důvodů.

B) Piny pro joints (osové drivery)

Vytváří je task.

```
joint.N.amp-ready    (bit IN)
```

Je to analogie existujících RT pinů:

- joint.N.amp-enable-out
- joint.N.amp-fault-in

Dává to konzistentní HAL jmenný prostor.

C) Piny pro spindles (vřetena)

```
spindle.N.amp-ready  (bit IN)
```

Také vytváří task.

==========

5. Sekvenční logika STARTING

==========

Shodli jsme se na přesném pořadí readiness kontrol:

1. Kontrola všech joint.N.amp-ready

- jeden po druhém
- v pořadí indexu N
- musí být všechny TRUE

2. Kontrola všech spindle.N.amp-ready

- stejný princip jako u joints

3. Kontrola machine.ready

- reprezentuje subsystémy stroje (hydraulika, pneumatika, PLC, chladicí jednotky…)

Po úspěchu:

- Task nastaví `machine.starting = FALSE` .
- Task přejde do stavu ON.

==========

6. Čtení a správa READY signálů

==========

Všechny HAL ready signály jsou **bit IN**. Task je čte periodicky během STARTING stavu.

Pokud je v konfiguraci volba `STARTING = FALSE`:

- Task žádné ready signály nečte.
- Piny se NEvytvářejí.
- Stroj přechází OFF → ON jako dnes.

==========

7. Logika timeoutu STARTING

==========

Pokud během STARTING:

- uplyne `STARTING_TIMEOUT` (pokud není -1)

Task:

1. Přepne STARTING → OFF
2. Sestaví seznam všech signálů, které byly FALSE
3. Operátorovi zobrazí pouze PRVNÍ selhaný signál
4. Terminál dostane seznam VŠECH selhaných signálů
5. Operátorovi vypíše i údaj o timeoutu (např. 10 s)

==========

8. Interrupce uživatelem (OFF během STARTING)

==========

Pokud uživatel:

- stiskne OFF
- GUI vyšle EMC_STATE_OFF
- halui.machine.off se aktivuje
- nebo dojde ke stavu ABORT

Task:

- přeruší STARTING
- přejde do OFF
- vypíše operátorovi hlášku: „Startování zrušeno uživatelem.“
- vypíše 1. selhaný ready signál
- v terminálu vypíše všechny selhané ready signály

==========

9. E-STOP a jiné prioritní stavy

==========

E-STOP má absolutní prioritu.

Pokud během STARTING dojde ke:

- fyzickému nouzovému tlačítku
- soft E-STOP (halui.machine.estop)

- safety PLC aktivaci ESTOP pinů

Task okamžitě:

- ukončí STARTING
- přepne do stavu EMC_TASK_STATE_ESTOP
- NEVYPISUJE ready signály (nemají smysl)
- vypíše pouze hlášku:

```
Startování přerušeno nouzovým zastavením (E-STOP).
```

To je správné a průmyslově bezpečné.

Další případy s vyšší prioritou:

- motion fault
- servo fault
- driver fault
- HAL error

Také nikdy neukazují ready-signály.

==========

10. Chybová hlášení – GUI vs. terminál

==========

Shodli jsme se na filozofii:

Operátor (GUI):

- zobrazit minimum
- pouze první selhaný signál
- a informaci o timeoutu nebo přerušení
- žádné přetěžování operátora

Servis / terminál:

- kompletní diagnostika
- výpis všech FALSE pinů
- jasný log, který lze předat výrobci stroje

Velmi profesionální přístup.

==========

11. Umístění logiky – TASK jako vlastník

==========

STARTING logika patří do:

task, nikoli:

- halui (UI),
- motion (RT),
- iocontrol (IO),
- ani do GUI.

Důvody:

- Task vlastní stav stroje
- Task řídí přechody mezi stavy
- Task čeká (má interní smyčku)

- Task má přístup k NML
- Task je jediná správná úroveň pro tento mechanismus

Motion je RT a nesmí čekat. Halui není povinné a nemůže být zdrojem machine-state. IOControl má jinou doménu (toolchanger, M-kódy).

==========

12. Varianty, které jsme výslovně zamítli

==========

A) Použít doménu `halui.machine.*`

Zamítnuto: halui je UI adaptér, volitelný modul, ne strojová logika.

B) Nechat IOControl generovat ready piny

Zamítnuto: IOControl má jasnou, úzce specializovanou doménu (toolchange, chills, lube).

C) Číst ready signály v motion

Zamítnuto: motion je realtime; nemá čekat; čistě RT vrstva.

D) Vytvořit nový task prefix `task.machine.*`

Možné, ale zamítnuto pro ergonomii a názvosloví → zvolili jsme kratší, čistší DOMÉNU `machine.*`.

==========

13. Možné budoucí rozšíření – machine.state

==========

Volitelný pin do budoucna:

```
machine.state   (s32 OUT)
```

Obsahoval by kompletní task state machine (ESTOP, OFF, STARTING, ON, RUNNING…).

Shodli jsme se, že:

- není nutné pro fázi 1
- není duplicita s halui.machine.is-on (protože halui.machine.is-on je jen boolean)
- machine.state je přesný a kompletní popis stavu stroje
- vhodné pro Gmoccapy, QtVCP a PLC-like integrace
- může být doplněn později

==========

14. Shrnutí hlavních architektonických zásad

==========

1. STARTING je volitelný stav mezi OFF a ON.

2. Aktivuje se přes INI `[TASK] STARTING=TRUE`.
3. HAL komunikace probíhá přes nové piny:
    - `machine.starting` (OUT)
    - `machine.ready` (IN)
    - `joint.N.amp-ready` (IN)
    - `spindle.N.amp-ready` (IN)
4. Sekvence readiness je pevná: joints → spindles → machine.
5. Timeout STARTING i uživatelské přerušení generují diagnostiku.
6. GUI dostává minimum hlášek, terminál maximum.
7. E-STOP, motion fault a safety stavy mají prioritu – bez ready diagnostiky.
8. Vše běží výhradně v tasku – architektonicky čisté.

==========

15. Co můžeme řešit v další etapě (implementace)

==========

1. Pseudokód celého STARTING automatu.
2. Umístění implementace v `emctask.cc`.
3. Přidání nového task state do `emctask.h`.
4. Inicializace nových HAL pinů.
5. Rozšíření NML hlášek pro GUI.
6. Úpravy v `emc/status` tak, aby GUI vidělo STARTING přes stat().
7. Dokumentace HAL pinů (manpage styl).
8. Testovací scénáře.

==========

Závěrečné potvrzení

==========

Ano — pokryli jsme celou architekturu integrace STARTING do LCNC:

- domény
- piny
- sekvenci
- INI konfiguraci
- stavový automat
- chování při chybách
- reakce na E-STOP a OFF
- výpisy do GUI a terminálu
- volitelnost
- kompatibilitu se stávajícími GUI a HAL
- a vše je architektonicky čisté, proveditelné a profesionální.

Pokud v této rekapitulaci něco nesouhlasí s vaší představou, okamžitě to opravíme. Pokud je vše v pořádku, můžeme začít s etapou návrhu implementace.