

What you've got in this sample

Two complete configs that only differ in how the carousel is sensed:

- `vmc_index_metric.ini` → uses `toolchange_index.hal` (index + pulse sensing)
- `vmc_graycode_metric.ini` → uses `toolchange_gray.hal` (4-bit Gray code)

Both run a Vismach simulated VMC with a carousel ATC and hook into Probe Basic's ATC UI (the "DynATC" widget).

The INI: where the big picture lives

Open e.g. `vmc_index_metric.ini`. The key sections:

[DISPLAY]

- `DISPLAY = probe_basic`
- `CONFIG_FILE = custom_config.yml` (Probe Basic options)
- `ATC_TAB_DISPLAY = 1` → show the **carousel** ATC tab
- `USER_ATC_BUTTONS_PATH = user_atc_buttons/` (optional custom ATC buttons)

[RS274NGC] (remaps + macros)

- `SUBROUTINE_PATH = macros_sim`

M6 is remapped to a G-code program with prolog/epilog:

REMAP=M6 modalgroup=6 prolog=change_prolog ngc=toolchange epilog=change_epilog

-

Helper remaps used by the UI and M6 logic:

M10 P... (move carousel best direction to P)
M11 / M12 (step CW / CCW)
M13 (home/reference carousel)
M21 / M22 (ATC out/unload / ATC in after load)
M24 (unclamp drawbar)
M25 (extend ATC to tool-change position)

- (They're all in `macros_sim/`)

[ATC] (parameters that drive the macros & UI)

- `POCKETS = 12` (carousel size shown in the UI and used by macros)
- `Z_TOOL_CHANGE_HEIGHT` (machine Z for clamp/unclamp)
- `Z_TOOL_CLEARANCE_HEIGHT` (safe machine Z for rotating the platter)
- `STEP_TIME` (purely UI animation pacing)

[HAL]

For the *index* variant you'll see:

```
HALFILE = vmc.hal           # motion, joints, basic sim plumbing
HALFILE = sim_vmc_metric.hal # Vismach+sim sensors/actuators
HALFILE = spindle.hal       # orient/pid demo
HALFILE = toolchange_index.hal # carousel component & ATC wiring (index/pulse)
HALFILE = sim_cannon.hal    # unrelated PWM demo
POSTGUI_HALFILE = probe_basic_postgui.hal # UI timer & spindle rpm hook
```

The HAL: what's wired to what

1) The carousel component

In **index** mode (`toolchange_index.hal`):

```
loadrt carousel pockets=12 dir=2 encoding=index
addf carousel.0 servo-thread
```

```
# analog → integer pocket request
addf conv-float-s32.2 servo-thread
net car-pos-req motion.analog-out-02 => conv-float-s32.2.in
net car-pos-s32 conv-float-s32.2.out => carousel.0.pocket-number
```

In **gray code** mode (`toolchange_gray.hal`):

```
loadrt carousel pockets=12 dir=2 encoding=gray num_sense=4
addf carousel.0 servo-thread
```

Why the converter? G-code “motion.analog-out-NN” pins are floats;
`carousel.0.pocket-number` is an `s32`. The config uses `conv-float-s32` to bridge them.

2) Motor drive & sensors (index mode mapping)

From `toolchange_index.hal`:

Outputs (to your actuators)

- `motion.digital-out-03` → `car-ccw`
- `motion.digital-out-04` → `car-cw`
- `motion.digital-out-00` → `arm-act` (ATC extend/retract valve)
- `motion.digital-out-02` → `tool-release` (drawbar solenoid)

Inputs (from your sensors)

- `motion.digital-in-03` → `index` (carousel home index)
- `motion.digital-in-04` → `pulse` (slot pulse)
- `motion.digital-in-01` → `arm-in`
- `motion.digital-in-00` → `arm-out`


- `motion.digital-in-02` → `tool-released`
- `motion.digital-in-05` → `tool-locked`

Carousel handshakes

- `carousel.0.enable` ↔ `car-enable` (you can gate it)
- `carousel.0.ready` ↔ `car-ready` (status to UI/logic)

Toolchange handshakes with LinuxCNC

```
net tool-prep-loop  iocontrol.0.tool-prepare  iocontrol.0.tool-prepared
net tool-change-loop iocontrol.0.tool-change  iocontrol.0.tool-changed
```

 The gray-code HAL wires 4 digital inputs to `carousel.0.sense-0..3` instead of `index/pulse`.

3) The sim “plant”

`sim_vmc_metric.hal` is a big Vismach simulation that **creates** virtual limit switches, ATC arm motion, a simulated carousel, gray-code generator, etc. In a real machine you’ll **remove/ignore** those simulated nets, and wire your real I/O instead.

The G-code macros (how M6 actually works)

The heavy lifter is `macros_sim/toolchange.ngc`, called by the M6 remap. Key ideas:

- **State variables** (persisted across sessions unless volatile noted)
 1. `#3989` — carousel homed flag (set by **M13**) (**volatile**)
 2. `#3990` — current pocket number (1..N)

3. `#3991` — tool number currently in spindle (0 = none)
 4. `#4001..#4024` — pocket→tool map (pocket *i* holds tool in `#[4000+i]`)
- **INI-driven limits**
 1. `#<number_of_pockets> ← [ATC]POCKETS`
 2. `#<atc_z_tool_change_height>, #<atc_z_tool_clearance_height> ← [ATC]...`
 - **Flow (simplified):**
 1. **Check:** valid task, homed, cutter comp off, etc. (`change_prolog` in `python/stdglue.py` injects `#<selected_tool>`, `#<current_pocket>`, etc.)
 2. **Find** the pocket that holds the requested tool (`#<next_pocket>`); also find **first empty pocket** to park a tool if spindle is non-empty.
 3. If a tool is in the spindle, run **M21** (ATC OUT + unload) or error if no empty pocket.
 4. Rotate the carousel to `#<next_pocket>` using **M10** (best direction).
 5. Extend ATC to cut position (M25/M21 path), orient spindle (via **M19** in `orientspindle.ngc`), **unclamp/clamp** with **M24/clamptool.ngc/unclamptool.ngc**, verify sensors with **M66** timeouts.
 6. Retract ATC home with **M22**, update `#3990/#3991/#400x`.
 7. Commit the change: **M61 Q...** and apply **G43 H...**
 - **UI hooks:** the macros sprinkle lines like `(DEBUG, EVAL[vcp.getWidget{"dynatc"}.store_tool{pocket, tool}])` to keep the Probe Basic ATC page in sync.

Note on M64/M65/M66 “P” numbers: these are **indices into motion’s digital out/in pins**. The sample sim HAL maps them to nets (see mapping above). On real hardware you’ll map those nets to your I/O. If you change the mapping, keep the “P” numbers and HAL nets in step.

Running the sim (quick sanity test)

1. Start LinuxCNC with `vmc_index_metric.ini` (or gray-code variant).
2. Home the axes.
3. In MDI:
 - `M13` → reference the carousel.
 - `M11 / M12` → nudge CW/CCW by one pocket.
 - `T3 M6` → full toolchange to tool 3 (make sure the tool table has T3).
4. Watch the ATC tab (DynATC) update pockets and spindle tool.

Diagnostics during testing:

`halcmd show pin carousel.0`

`halcmd show pin motion`

Adapting this sample to a real machine (incl. EtherCAT)

1. **Pick your sensing style**
 - **Index + pulse:** single “home index” plus one pulse per pocket.
 - Use `toolchange_index.hal`, `encoding=index`, wire:
 - `carousel.0.sense-0` ⇐ index/home sensor
 - `carousel.0.sense-1` ⇐ pocket pulse sensor

- **Gray code (absolute):** 4 inputs give pocket number directly.
 - Use `toolchange_gray.hal`, `encoding=gray num_sense=4`, wire:
 - `carousel.0.sense-0..3` \Leftarrow gray bits (0..3)

2. Map the nets in `toolchange_*.hal` to your I/O

- Keep the **motion.digital-out** and **motion.digital-in** names used by the macros (so M64/M66 P numbers keep working).
- Then connect those **nets** to your hardware pins.

Example skeleton with EtherCAT (names will vary — check with `halcmd show pin lcec`):

```
loadusr -W lcec_conf ethercat.xml
loadrt lcec
```

Outputs

```
net car-cw      => lcec.0.2.dout-0
net car-ccw     => lcec.0.2.dout-1
net arm-act     => lcec.0.3.dout-0
net tool-release => lcec.0.3.dout-1
```

Inputs

```
net index      <= lcec.0.4.din-0
net pulse      <= lcec.0.4.din-1    # index mode only
net arm-in     <= lcec.0.4.din-2
net arm-out    <= lcec.0.4.din-3
net tool-released <= lcec.0.4.din-4
net tool-locked <= lcec.0.4.din-5
```

Keep the conv-float-s32 bridge for pocket requests

```
net car-pos-req motion.analog-out-02 => conv-float-s32.2.in
net car-pos-s32 conv-float-s32.2.out => carousel.0.pocket-number
```

3. EtherCAT pin names depend on your XML/slaves. The **pattern** is what matters: connect the existing *nets* to your hardware pins.
4. **Set mechanical parameters**

- `[ATC]POCKETS`
- `[ATC]Z_TOOL_CHANGE_HEIGHT` (machine Z to clamp/unclamp a tool)
- `[ATC]Z_TOOL_CLEARANCE_HEIGHT` (safe rotate height)
- If your drawbar/ATC logic differs, adjust the macros in `macros_sim/` (they're just G-code).

5. **Spindle orientation** (if you need it)

- The sample `spindle.hal` demonstrates **M19** orientation with a PID + `orient` component. For a real encoder/drive, wire your actual encoder velocity/position, and the orient lock output.


6. **Handshake stays the same**

- Keep `iocontrol.0.tool-prepare/tool-prepared` and `tool-change/tool-changed` nets; the remap epilog commits the toolchange via `emccanon` calls.

Quick “cheat sheet” of signals used by the macros

- **Outputs** (set/clear with `M64/M65 Pn`)
 - `P0` — ATC extend (carousel OUT) valve (energize = out, de-energize = in)
 - `P1` — used in macros for “ATC OUT position check” (see note below)
 - `P2` — drawbar **unclamp** solenoid
 - `P3/P4` — carousel step (CW/CCW) in `M11/M12/M13` flows
- **Inputs** (wait with `M66 Pn L... Q...`)

- P0 — “ATC IN” position sensor
- P1 — “ATC OUT” position sensor
- P2 — **tool released** sensor (unclamped)
- P5 — **tool locked** sensor (clamped)
- P4 — index/pulse (homing/stepping checks)

 **Important:** The exact P↔sensor mapping depends on how you connect `motion.digital-in-*` in your HAL. The *sample sim* maps those pins one way; your real machine can map them differently — just keep the HAL nets (`arm-in/out`, `tool-locked/released`, `index/pulse`) consistent with the macros’ expectations.

How to switch between Index and Gray code

- Use the corresponding INI that sources the right HAL file:
 - **Index:** `vmc_index_metric.ini` + `toolchange_index.hal`
(`encoding=index`)
 - **Gray:** `vmc_graycode_metric.ini` + `toolchange_gray.hal`
(`encoding=gray num_sense=4`)
 - Wire sensors accordingly.
 - `[ATC]POCKETS` must match your real carousel.
-

Common pitfalls (and quick fixes)

- **Carousel doesn't move / UI not updating**
 - Check `carousel.0.enable` is true.
 - `halcmd show pin carousel.0` and confirm `.ready` toggles.
 - Verify the analog→s32 bridge to `pocket-number` is connected.
- **M6 aborts with “not homed”**
 - Run `M13` first; it sets `#3989` and aligns `#3990`.
- **Wrong pocket numbers (Gray code)**
 - Swap your bit wires until the pocket number increments in the right order (Gray bit order must match `sense-0..3`).
- **M64/M66 P numbers “don't match my wiring”**
 - Leave the macros alone; remap **HAL nets** to your hardware pins so that `motion.digital-*` indices line up.