Introduction to DPLL subsystem

Vadim Fedorenko
Production Engineer

Meta

Agenda

01 Overview of DPLL devices

02 Protocol details

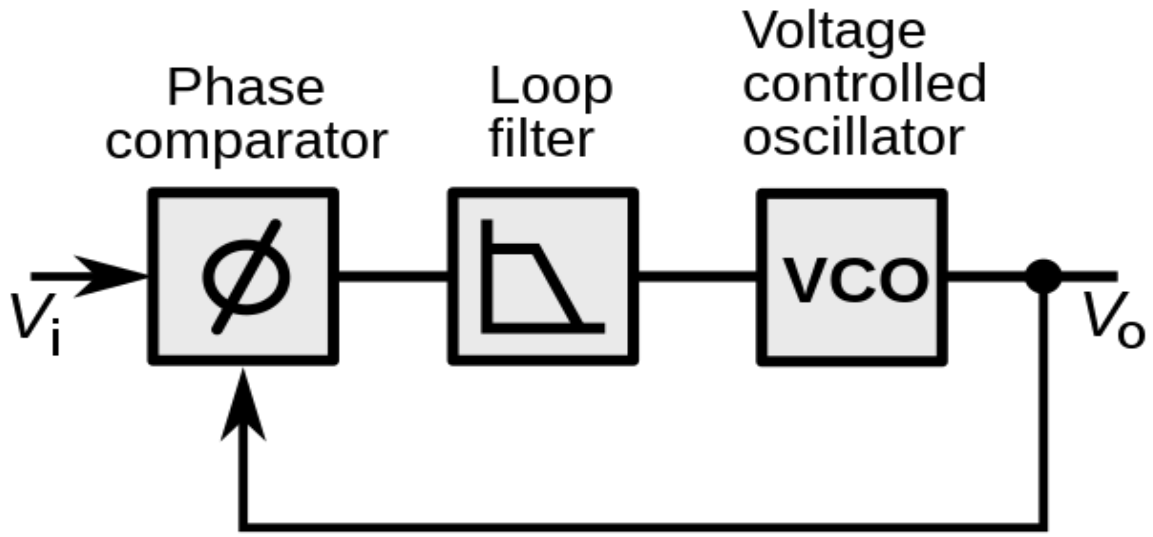
03 Implementation in drivers

04 Next steps

05 Credits

DPLL stands for Digital Phase-Locked Loop

A Phase-Locked Loop is a system which generates output signal which phase is related to the phase of an input signal.



The Digital here stands for using numerically controlled oscillator (NCO) and using digital comparator and filter.

Overview of DPLL devices

- The feedback path usually have frequency divider/multiplier to provide new output frequency with the phase aligned to the input signal.
- Different types of signals can be used as an input.
- The automatic fallback may be implemented if the device supports input prioritization option.
- Signal muxes may be used to extend the amount of inputs and outputs of DPLL devices

PROTOCOL DETAILS

NetLink transport

1. DPLL device object
    - Identification attributes
    - Mode (Auto/Manual) and type (PPS/EEC*) configuration attributes
    - Monitoring attributes (lock status, phase offset and temperature)

*Ethernet equipment slave clock G.8262/Y.1362

NetLink transport

2. Device pin object
   - Identification and connection attributes
   - Label attributes
   - Pin direction attributes
   - Pin type attribute (external, internal, SyncE ethernet port ..)
   - Pin state attribute (connected/disconnected or selectable)
   - Pin frequency attribute (some constants but variable frequency is also possible)
   - Pin capabilities attributes
   - Connection attributes (pin to device, pin to pin)

NetLink transport

3. Details are in the linux tree
    - NetLink primitives are auto-generated
    - YAML schema is in Documentation/netlink/specs/dpll.yaml
    - API documentation is in Documentation/driver-api/dpll.rst
    - Implementation examples are in drivers

Drivers API

Operations structure for DPLL device:

```
struct dpll_device_ops {
   .mode_get();          /* mandatory */
   .mode_supported();
   .lock_status_get();   /* mandatory */
   .temp_get();
};
```

Drivers API

Operations structure for DPLL device:

```
struct dpll_device_ops {
   .mode_get();          /* mandatory */
   .mode_supported();
   .lock_status_get();   /* mandatory */
   .temp_get();
};
```

Operations structure for PIN object:

```
struct dpll_pin_ops {
   .frequency_set();
   .frequency_get();
   .direction_set();
   .direction_get();     /* mandatory */
   .state_on_pin_get();  /* mandatory */
   .state_on_dpll_get(); /* mandatory */
   .state_on_pin_set();
   .state_on_dpll_set();
   .prio_get();
   .prio_set();
};
```

Drivers ready

- OCP TAP driver (ptp_ocp) for Time Card.
  - Support for PPS mode only
  - Simple configuration of 4 external pins
  - Simple monitoring of lock status

Drivers ready

- OCP TAP driver (ptp_ocp) for Time Card.
    - Support for PPS mode only
    - Simple configuration of 4 external pins
    - Simple monitoring of lock status
- nVidia mlx5 driver (implemented by Jiri Pirko)
    - Support for EEC* mode
    - SyncE is configurable on ethernet port

*Ethernet equipment slave clock G.8262/Y.1362

Drivers ready

- OCP TAP driver (ptp_ocp) for Time Card.
    - Support for PPS mode only
    - Simple configuration of 4 external pins
    - Simple monitoring of lock status
- nVidia mlx5 driver (implemented by Jiri Pirko)
    - Support for EEC* mode
    - SyncE is configurable on ethernet port
- Intel ice driver (implemented by Arkadiusz Kubalewski)
    - PPS/EEC modes
    - Multiple DPLL devices on physical board
    - External pins are connected directly to DPLL
    - SyncE is configurable on ethernet ports through mux devices

*Ethernet equipment slave clock G.8262/Y.1362

Testing

- RFC patches from Michal Michalik (Intel)
    - Core framework
    - No need for specific hardware
    - No need for special system architecture
    - Simple test cases are covered

Credits

- Arkadiusz Kubalewski, Michal Michalik, Milena Olech from Intel
  - DPLL pins as separate objects
  - Complex configurations with MUX devices
  - ICE driver support
- Jiri Pirko from nVidia
  - Connection between netdevice and DPLL pin
  - MLX5 driver support

Credits

- Arkadiusz Kubalewski, Michal Michalik, Milena Olech from Intel
  - DPLL pins as separate objects
  - Complex configurations with MUX devices
  - ICE driver support
- Jiri Pirko from nVidia
  - Connection between netdevice and DPLL pin
  - MLX5 driver support

Thank you!

Questions?

Meta