Contribution to run control discussion:

State machine and synchronous commands

Irakli Mandjavidze

Irfu, CEA Saclay
Gif-sur-Yvette, 91191
France

Internal

6/Jan/2025

Some proposals for run control a state machine

- A state machine that brings FE (ASIC) from “on” state to “running” state through several intermediate states
  - → For “lower” level states, transition requires asynchronous commands (*e.g.* configuration)
  - → For “higher” level states, transition requires synchronous commands (*e.g.* data taking)

- An example of states
  - → On : after power-up
    - ▪ Attempts to validate clocks and serial IOs
  - → ClockOk → IoLinksOk : input and derived clocks are validated, communication with DAQ established
    - ▪ Attempts to validate rest of the logic
  - → Initial : all logic is up and running with its default configuration
    - ▪ Waits for successive asynchronous commands for configuration

Proper for each sub-system

Example only
Reality much more complex
Not discussed

  - → Ready : configuration done, ready to take data
    - ▪ Waits for successive synchronous commands
  - → Running : data taking in progress
    - ▪ Data taking in progress; re-configuration is forbidden
    - ▪ Waits for synchronous commands either to return to Ready state or to go to “Pause” state
  - → Pause : data taking is suspended
    - ▪ Waits for synchronous commands
      - To recover from errors and resynchronize
      - To resume data taking or to stop data taking

Standardized throughout the experiment

Proposal only
Discussed in what follows

  - → Error : for whatever reasons
    - ▪ May requires a “heavy” intervention, but an attempt can be done to recover quickly

Proper for each sub-system
Requires cooperation with central SM

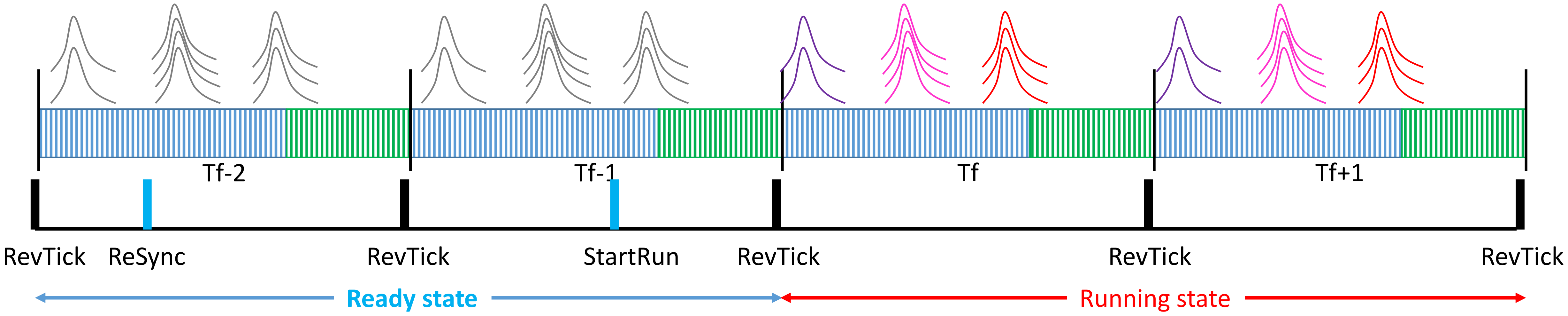
Some proposals : run control synchronous commands

- The aim is to ensure synchronous transition to/from running (data taking) state of entire acquisition system
  - → With a possibility of quick re-synchronization or error recovery
- The run control commands have mostly broadcast nature
- A list of few possible run control commands
  - → StartRun_Cmd
  - → StopRun_Cmd
  - → ReSync_Cmd
  - → RevTick_Cmd
  - → Auxiliary : Pause_Cmd and Resume_Cmd : suspend and resume data taking
  - → Other auxiliary commands can be envisaged, typically to support triggered readout : *e.g.* RstEvId_cmd
    - ■ Not discussed
- Certain synchronous commands may require an acknowledgment packet to be sent to a system supervisor
  - → Feedback to supervisor on the state of sub-systems
  - → Example can be commands that result to the state change, such as Start, Stop, Pause, Resume

Some proposals : Start of run

- The frontend (ASIC) is in the Ready state
  - → Configuration done, ready to take data
  - → Waits for a sequence of successive synchronous commands

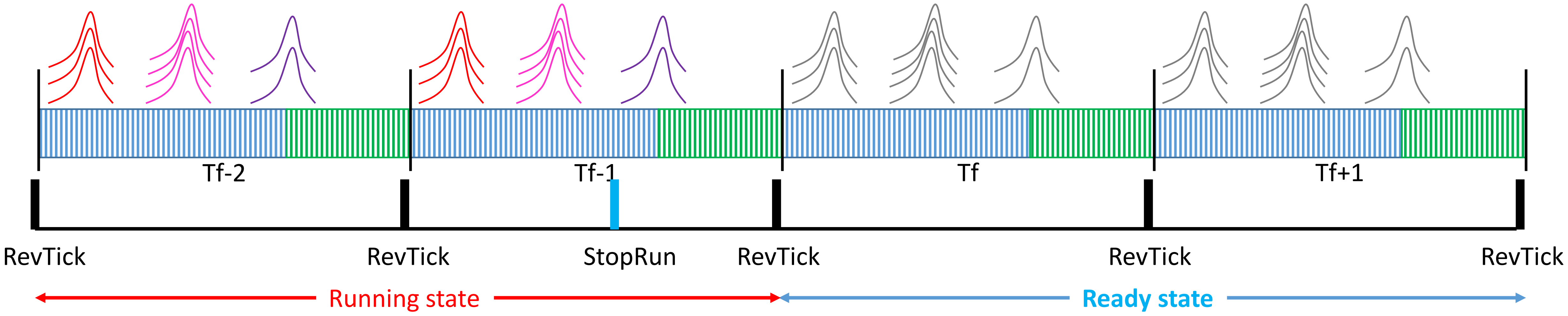
- Assume synchronization of sub-systems has not been done yet, the sequence can be
  - → ReSync_Cmd : a kind of “warm” reset, configuration registers are not affected
    - ■ Makes sure all current data in pipelines and in FIFOs are emptied (or possibly processed and sent out)
    - ■ Performs necessary actions on resynchronization if any
    - ■ Resets local timestamp and time frame counters to predefined values
  - → StartRun_Cmd : indicates that data taking must start after next RevTick_Cmd
    - ■ Gives enough time to subsystem electronics to do last steps (if any needed) before data taking actually begins
  - → RevTick_Cmd : frontend (ASIC) goes to Running state and starts to deliver acquired data



Some proposals : End of run

- The frontend (ASIC) is in Running state
  - → Needs a sequence of successive synchronous commands to stop data taking

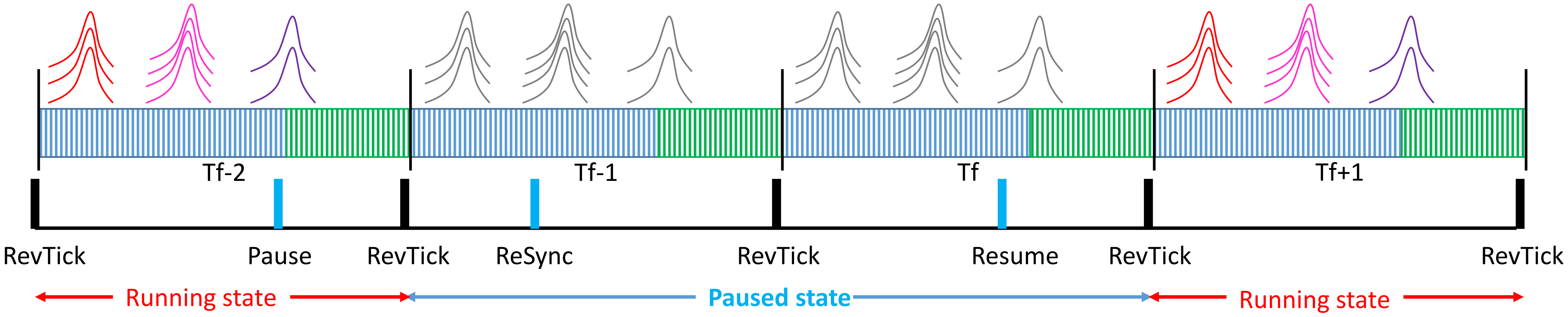
- The sequence can be
  - → StopRun_Cmd : indicates that data taking must stop after next RevTick_Cmd
  - → RevTick_Cmd :
    - Frontend (ASIC) does not accept new data
    - Finishes data delivery of the elapsed time frame
    - Goes to Ready state



Some proposals : Pause and Resume

- An attempt of quick recovery from some run-time errors can be envisaged with Pause and Resume commands
  - → Upon reception of Pause_Cmd the system goes to Paused state following the next RevTick_Cmd
    - ■ Data from elapsed time frame delivered
  - → Healthy sub-systems are kept ready to restart data taking
  - → Faulty sub-system attempts to recover
    - ■ Warm reset, link synchronization, whatever
  - → If faulty sub-system becomes healthy, restart data taking sending a sequence of synchronous commands
    - ■ ReSync_Cmd, (RstEvId_Cmd)
  - → Upon reception of Resume_Cmd the system goes to Running state following the next RevTick_Cmd

- Pause / resume commands can be handy for system-wide monitoring inducing very little dead-time



Summary on run control state machine

- Need to know ePIC DAQ state machine
  - → Understand its influence on frontend design
  - → Make sure ePIC DAQ takes into account the needs of the MPGD frontends and readout

- Need to know synchronous commands
  - → Defined by ePIC DAQ an obligatory set of commands to obey to and a protocol to follow
    - ■ *e.g.* broadcast commands like StartRun, EndRun, etc.
  - → A set of commands needed by subsystem readout
    - ■ *e.g.* Multicast commands like Calib; support for coupled sync commands like ArmPulser followed by Calib

- Need to know command distribution details
  - → 40 MHz / 100 MHz boundaries
  - → Convention on command timing
    - ■ *e.g.* are certain commands bound to particular bunch crossing IDs?

- What was shown is (possible) example only
  - → Ready / Running / Pause state transitions may not be as rigid and strictly synchronous to EIC revolutions
    - ■ A new time frame command can be used bounding beginning and ending of data taking to ePIC “Time Frames”

- But whatever decided, we need a document that sets the ePIC DAQ rules