FIR IP (FIR_DA_full_v4, updated version of the original FIR DA IP)

Figure 1 shows the FIR DA IP with its I/Os and parameters. 'sclr' clears the input register chain of the 1D FIR core. 'rst' clears the shift register for 'v' (to protect it after PR).
'v' is output valid. Note that usually 'v' is a delayed version of 'E', except when 'E' and 'sclr' are asserted at the same time.
It also allows for the **antisymmetric mode** (SYMMETRY = AYES).

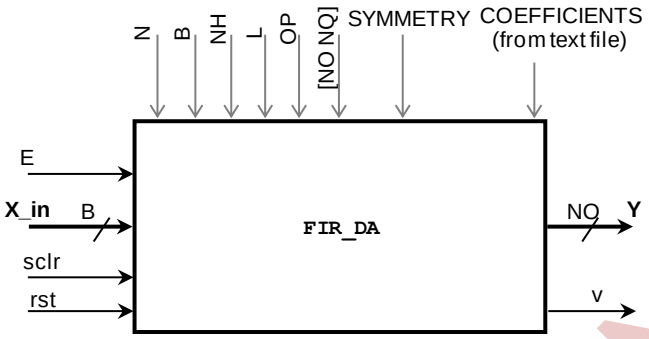


Figure 1. FIR Filter IP

PLB INTERFACE (plb_ip_rlrH_v1)

Fig. 2 shows the PLB interface for the real filter IP. We support 3 I/O cases: B=NO=8, B=NO=16, and B=8,NO=16 (in the project plb_ip_rlrH_v1, we actually used complex_fir_da core but with the <real input, real coefficients> option, this is effectively the same as the fir_da core).
This project supports V4 and V6 (FIFO primitives are different in this case).

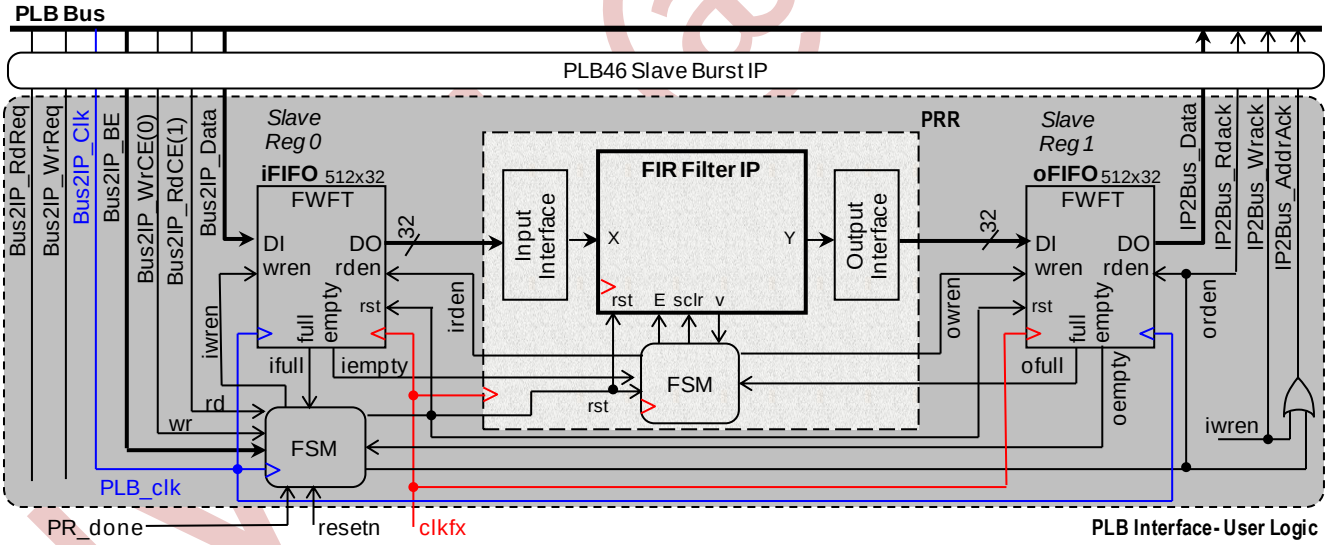


Figure 2. PLB interface for complex filter IP

FIFO16: Virtex-4 FIFO primitive. iFIFO and oFIFO have different clocks (dual clock FIFO). The following needs to be set: FWFT mode (no output register). Primitive: `FIFO16` → 18Kb FIFO (512x36, `PLBW=32`)
The FIFO has a `rst`, that is high for at least three `rdclock` cycles, `rden` held low for 4 cycles before `rst` is high and it must remain low during the reset cycle.
Fig. 3 shows the 3 I/O cases. These 3 cases were successfully simulated for lena(CIF), nr=2, N=32, NH=16, L=4, symmetric and anti-symmetric. All modes worked STREAM, IMG, FILT, CONV (see section FSM @ clkfx)

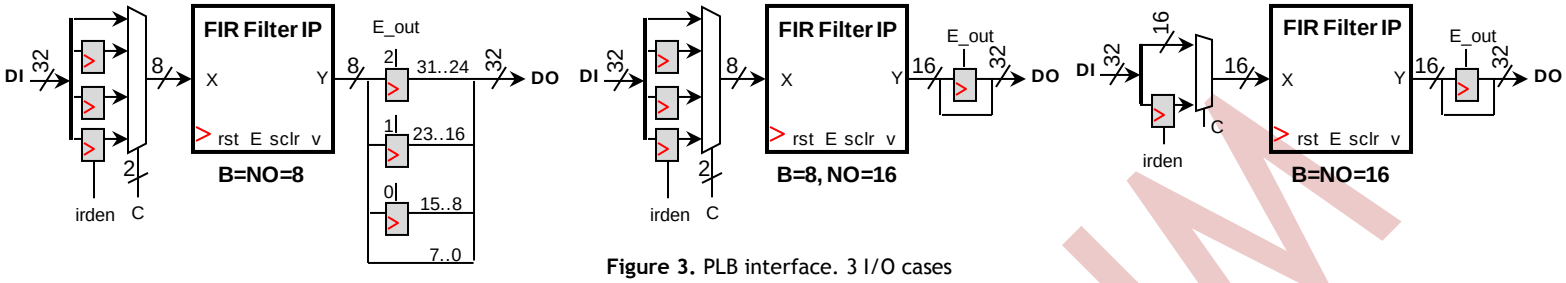


Figure 3. PLB interface. 3 I/O cases

`rst`: resets the register 'v'. `sclr`: It ONLY clears the input register chain.
* ML405: the external reset is low-level ('resetn')

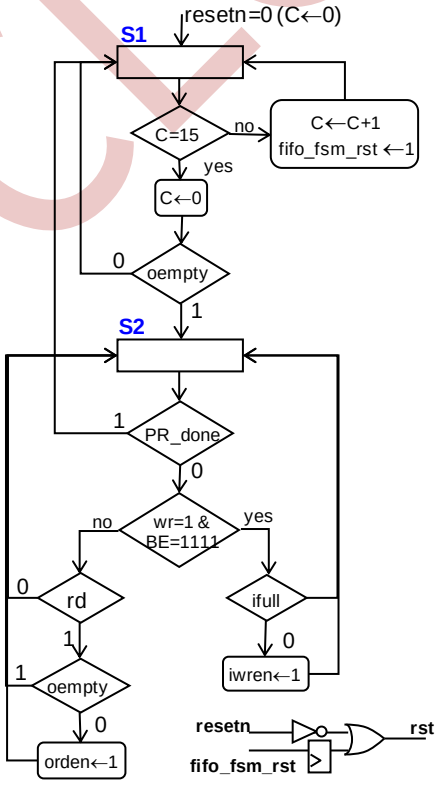
DYNAMIC REGION (PRR) I/O:

`dyn_inv(31..0) ← iFIFO_DO`	`oFIFO_DI ← dyn_outv(31..0)`
`dyn_inv(32) ← ofull`	`owren ← dyn_outv(32)`
`dyn_inv(33) ← iempty`	`irden ← dyn_outv(33)`
`dyn_inv(34) ← rst`	

FSM @ PLB_CLK

- FIFOs have to be reset prior to usage for at least 3 read/write clock cyles. If we use 16 cycles @ 100 MHz, then the minimum clkfx is $16 \times 10ns/3 = 53.33ns \rightarrow 18.75MHz$.

Figure 4 shows the FSM at PLB_clk. The register to 'fifo_fsm_rst' is just to avoid glitches (this is not a big deal, it was done to avoid simulation problem as a reset to a FIFO has to be clean).



FSM at Bus2IP_Clk

Figure 4. FSM at PLB_clk. 'C' represents a counter.

FSM @ CLKFX

There are 4 modes: STREAM, IMG, FILT, CONV. Each mode requires a different State Machine.

$NWI = \frac{32}{B}$ → # of input real pixels contained in a 32-bit word

$NWO = \frac{32}{NO}$ # → # of output real pixels contained in a 32-bit word

Fig. 5 shows the real pixel representation. If B=8, we can fit four real pixels in a 32-bit word. If B=16, we can only fit 2 real pixels.

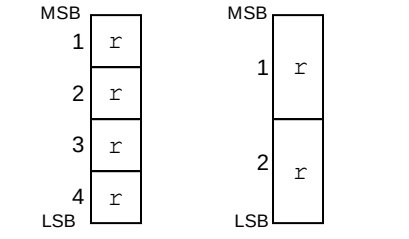


Figure 5. Real pixels' arrangement

Fig. 6 shows the way to generate '`owren`' for the modes STREAM and FILT. It is a little bit different for each I/O case: B=NO=8, B=8, NO=16, and B=NO=16. Note that 'irden' is usually activated along with 'E', and since 'v' is usually a delayed version of 'E', then most of the time 'v' would be the same as a delayed version of irden (by REG_LEVELS). However, this is not always the case, so we must refrain from trying to think that 'v' could be obtained from this shift register in Figure 6.

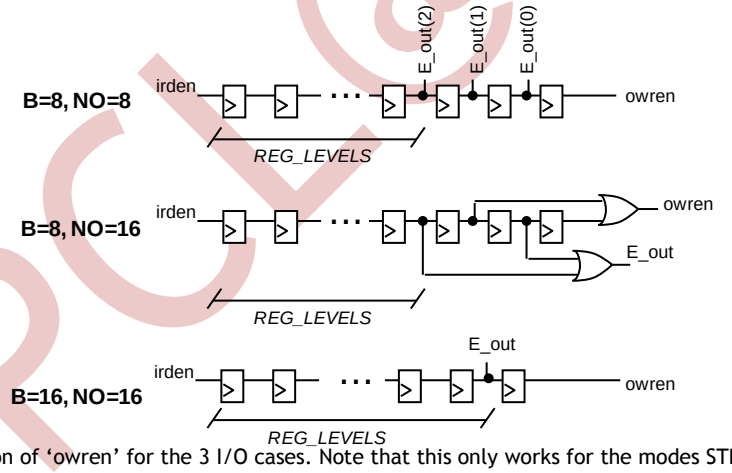


Figure 6. Generation of 'owren' for the 3 I/O cases. Note that this only works for the modes STREAM and FILT

Fig. 7 shows the timing diagram of the aforementioned cases (only STREAM and FILT) for the state machines (to be fully described later).

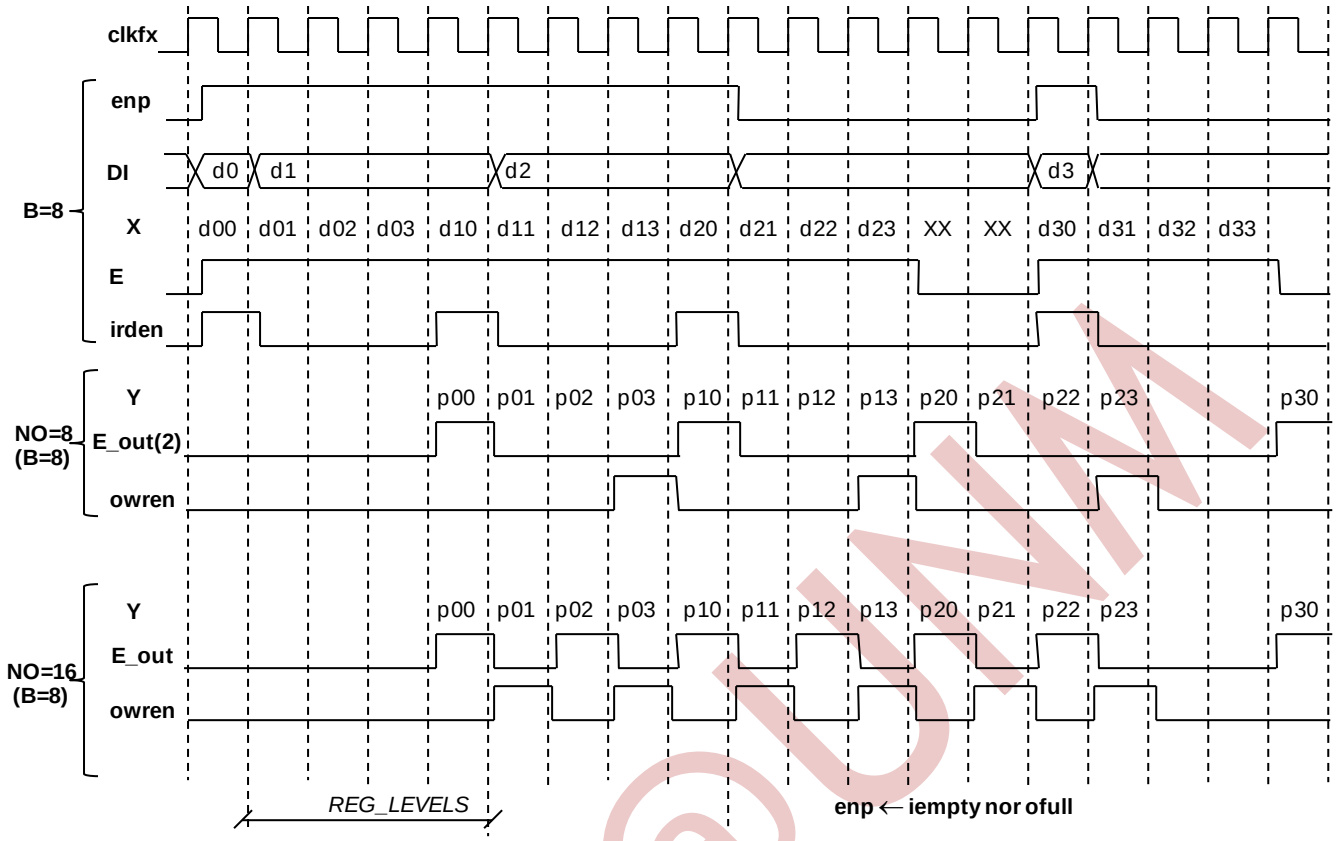


Figure 7. Timing diagram for modes STREAM and FILT. The generation or 'irden' is carried out in the State machines (to be explained later)

FSM for the mode STREAM

Fig. 8 shows the FSM for the STREAM mode. Note that 'sclr_regs' is the same signal as '`sclr`'. The choice of '`sclr_regs`' is so that we are actually clearing the register chain inside the FIR filters. In the rIrH case, we are confident that NWI>1.

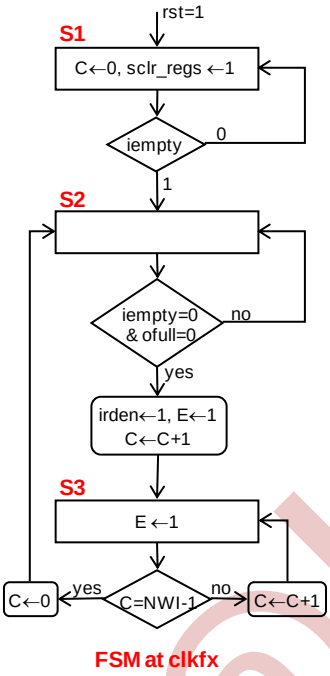


Figure 8. FSM for the STREAM mode.

FSMs for the mode IMG

Fig. 9 shows the FSM for the IMG mode. This FSM controls the input side. '`sclr_regs`' only clears the register chain inside the FIR filter (it does not clear other registers). The only constraint is that *NX* has to be a multiple of *NWI*. (In the rIrH case, we are confident that NWI>1 and NWO>1)

Since in this mode, the filter runs on a stream-by-stream basis, we need to clear the register chain after a stream of length *NX* has been processed. We have to let the last zero-valued pixel enter the chain, that is why we clear the chain after the last zero-valued pixel enters the chain. This is why we need a new state just for clearing the register.

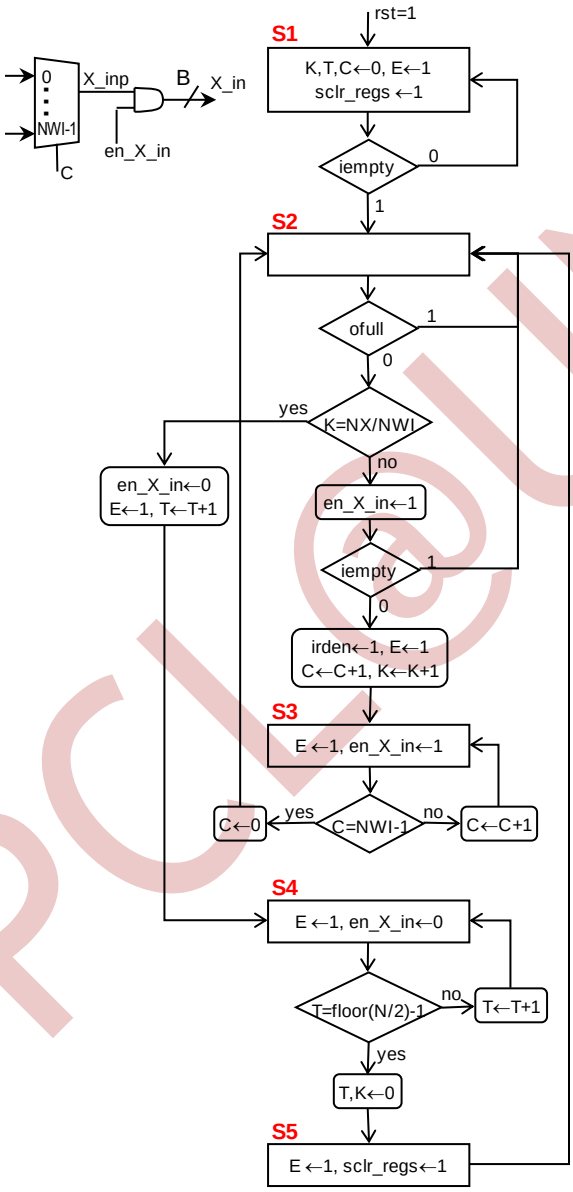


Figure 9. FSM for the IMG mode (input side)

Fig. 10 shows the FSM for the output side. '`owren`' is controlled by an FSM. The '`v`' output is employed. The case NWO=4 and NWO=2 are shown.

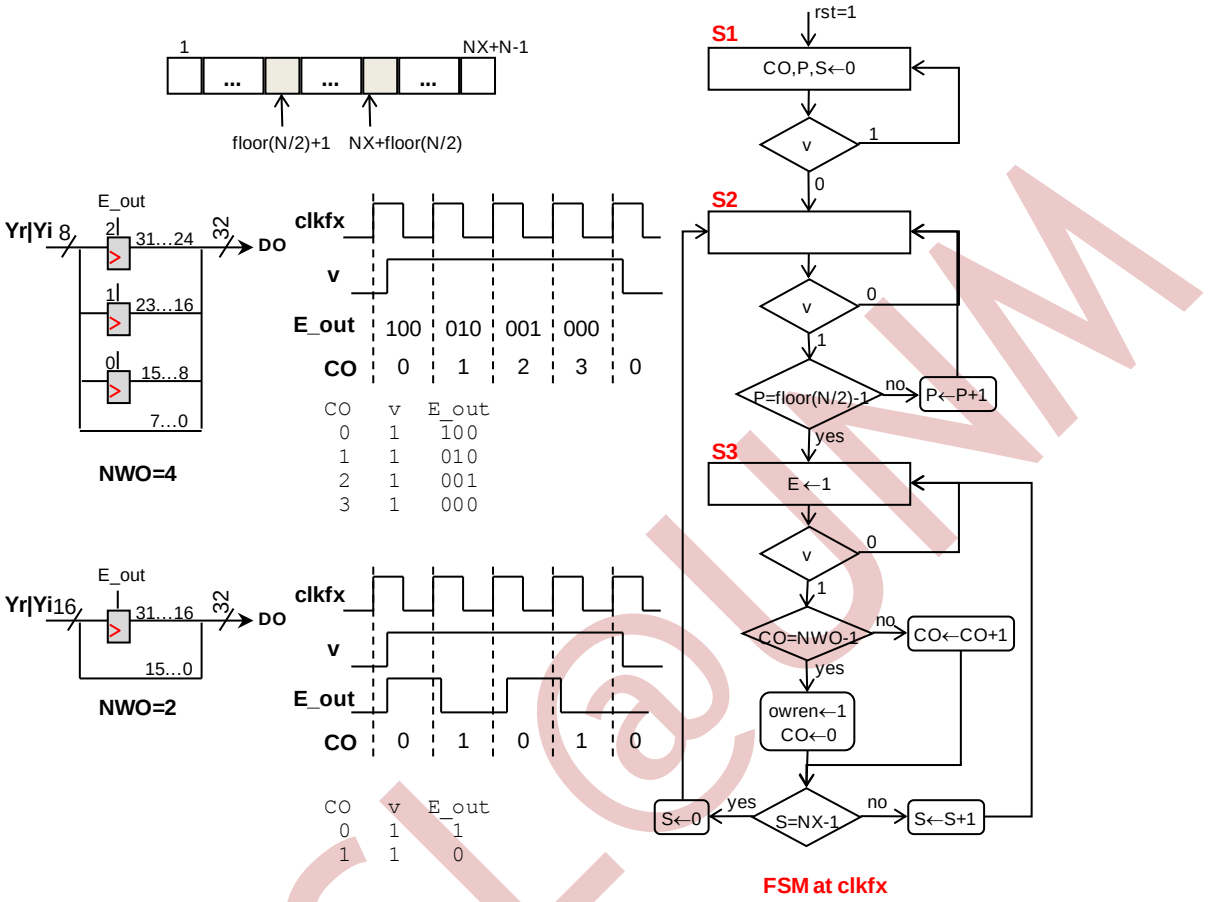


Figure 10. FSM for the IMG mode (output side)

FSM for the mode FILT

Fig. 11 shows the FSM for the FILT mode. '`sclr_regs`' only clears the register chain inside the FIR filter (it does not clear other registers). The only constraint is that *NX* has to be a multiple of *NWI*. In the rIrH case, we are confident that NWI>1.

Since in this mode, the filter runs on a stream-by-stream basis, we need to clear the register chain after a stream of length *NX* has been processed. We have to let the last zero-valued pixel enter the chain, that is why we clear the chain after the last zero-valued pixel enters the chain. This is why we need a new state just for clearing the register.

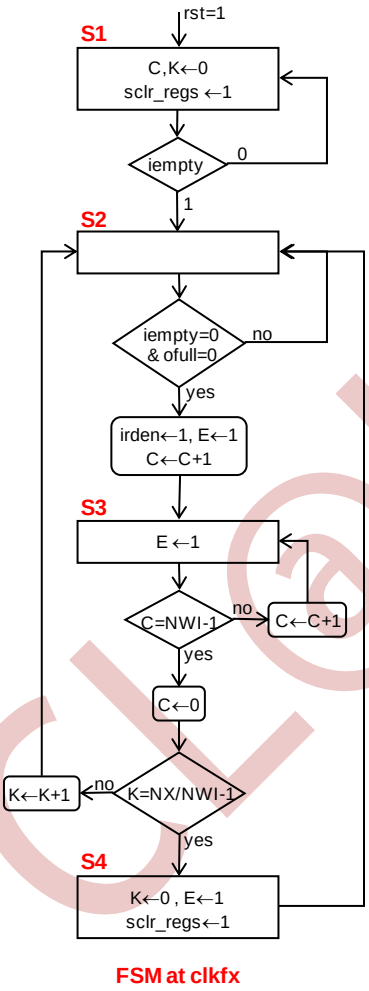


Figure 11. FSM for the FILT mode

FSMs for the mode CONV

Fig. 12 shows the FSM for the CONV mode. This FSM controls the input side. '`sclr_regs`' only clears the register chain inside the FIR filter (it does not clear other registers). The only constraint is that *NX* has to be a multiple of *NWI*. In the rIrH case, we are confident that NWI>1 and NWO>1.

Since in this mode, the filter runs on a stream-by-stream basis, we need to clear the register chain after a stream of length *NX+N-1* has been processed. We have to let the last zero-valued pixel enter the chain, that is why we clear the chain after the last zero-valued pixel enters the chain. This is why we need a new state just for clearing the register.

`E_out` is generated in the same way as it was generated for the mode IMG.

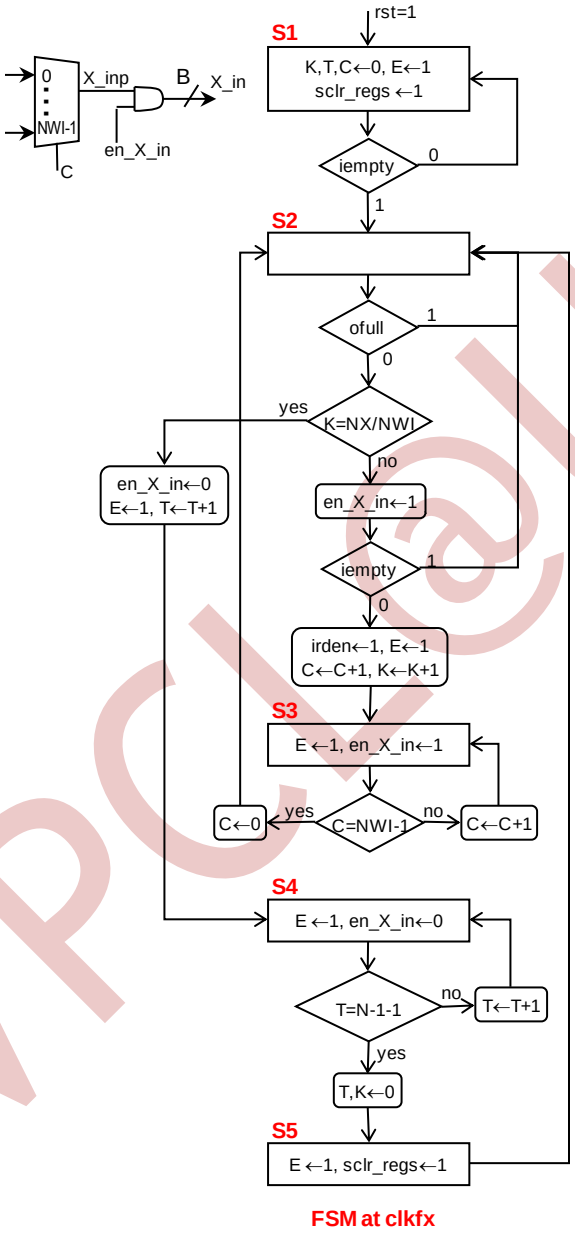


Figure 12. FSM for the CONV mode (input side)

Fig. 13 shows the FSM for the output side. '`owren`' is controlled by an FSM. The '`v`' output is employed.

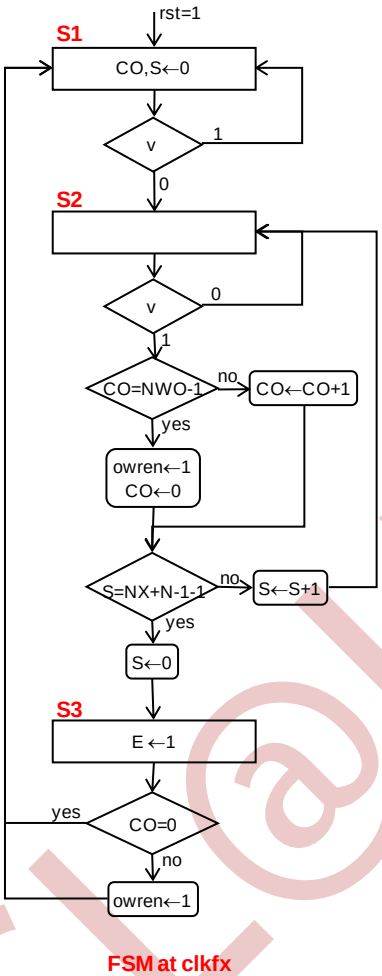


Figure 13. FSM for the CONV mode (output side)