COMPLEX FIR FILTERING

1D COMPLEX CONVOLUTION

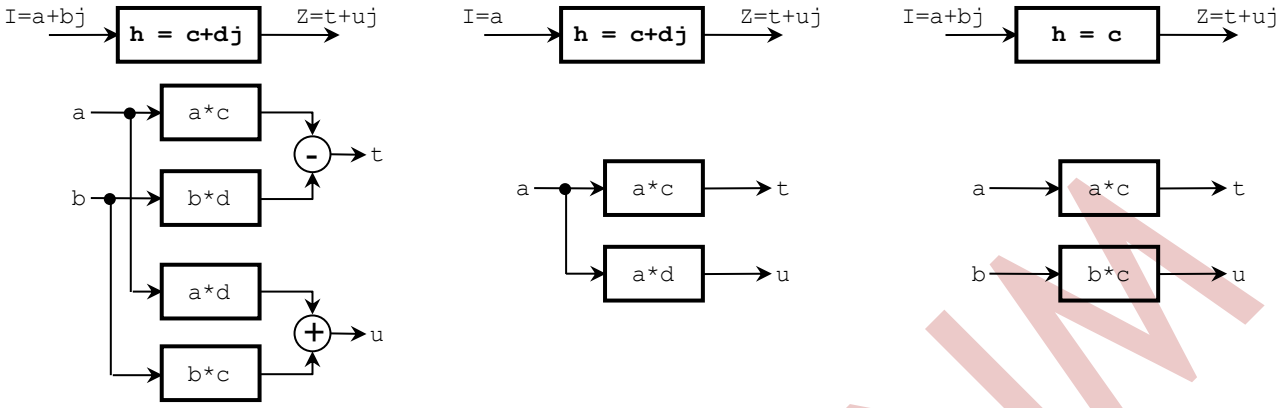


Figure 1. 1D complex convolution. Three cases: complex input, complex coefficients; real input, complex coefficients; complex input, real coefficients

Complex input, complex coefficients:
$(a+bj)*(c+dj)=(a*c-b*d)+(a*d+b*c)j$

Real input, complex coefficients:
$(a)*(c+dj)=(a*c)+(a*d)j$

Complex input, real coefficients:
$(a+bj)*(c)=(a*c)+(b*c)j$

2D SEPARABLE COMPLEX CONVOLUTION

This is performed on images. First we apply 1d convolution to the rows, then we transpose the row-filtered image, and finally we apply 1d convolution to the columns.

To understand the issues associated with 2d separable convolution, let's assume that we are just applying 2d convolution to one value. Then, 2d separable convolution is performed by applying 1d convolution followed by another 1d convolution.

$I=a+bj \quad W=h*v, h=c+dj, v=f+gj$

$Y=I*W=I*(h*v)=(I*h)*v$

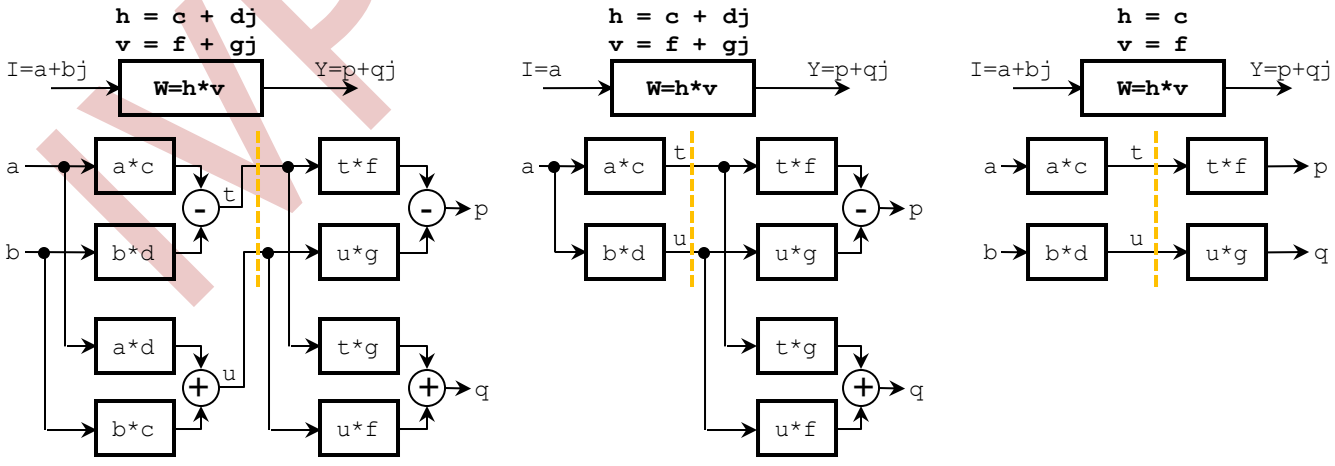


Figure 2. 2D separable convolution . Three cases: complex input, complex coefficients; real input, complex coefficients; complex input, real coefficients

1D COMPLEX FILTER IP

CONSIDERATIONS:

- The 1D complex FIR IP has four modes:
  - rIcH: real input, complex coefficients
  - cIrH: complex input, real coefficients
  - cIcH: complex input, complex coefficients
  - rIrH: real input, real coefficients: This is just the real filter

- The case complex input, complex coefficients (1D) adds an extra bit (because of the addition/subtraction). So, the IP performs the output truncation scheme (OP=0,1,2) on the outputs of the adder and subtractor rather than on the output of the filter.
  The maximum number of bits required for each output (real/imaginary) is: $Li\_FIR = NH + B + \lceil \log_2(N+1) \rceil$ (one more than in the real case). The output format then results: $[Li\_FIR \quad NH + B - 2]$
  Derivation:
  $-2^{B-1} \le x[k] \le 2^{B-1} - 1 \quad (B\ bits), \qquad -2^{NH-1} \le h[k] \le 2^{NH-1} - 1 \quad (NH\ bits)$
  Then we perform: $y = \sum_{k=0}^{N-1} x[k] \times h[k]$ (summation of N products) twice and add the results. We do this for each component (real and imaginary)
  We see that: Largest positive value: $\left(-2^{NH-1}\right)\left(-2^{B-1}\right)N \times 2 = 2^{NH+B-1}N$
  Largest negative value: $\begin{cases} \left(-2^{NH-1}\right)\left(2^{B-1}-1\right)N \times 2 = -2^{NH+B-1}N + 2^{NH}N, if\ NH \le B \\ \left(-2^{B-1}\right)\left(2^{NH-1}-1\right)N \times 2 = -2^{NH+B-1}N + 2^{B}N, if\ B < NH \end{cases}$
  → $-2^{NH+B-1}N + \min\left(2^{NH}N, 2^{B}N\right) \le y \le 2^{NH+B-1}N \equiv -2^{NH+B-1}N < y \le 2^{NH+B-1}N$

  i) N is a power of 2: $N = 2^n$ → $-2^{NH+B-1+n} < y \le 2^{NH+B-1+n}$
  Recall: $If\ \#bits of\ v = x \rightarrow -2^{x-1} \le v \le 2^{x-1} - 1 \quad (\gamma)$
  If $-2^{NH+B-1+n} < y \le 2^{NH+B-1+n} - 1$, then 'y' would require $NH + B - 1 + n + 1$ bits. However, we miss it by one LSB, and as a result, we need $NH + B - 1 + n + 2 = NH + B + 1 + \log_2 N$ bits.

  ii) N is not a power of 2:
  Let's define *Np* to be the immediate power of 2 after N → $Np/2 < N < Np$ → $\log_2(Np/2) < \log_2 N < \log_2 Np$
  Then, we conclude that $\lceil \log_2 N \rceil = \log_2 Np$, and $Np = 2^{\lceil \log_2 N \rceil}$. Also: $2^{\lceil \log_2 N \rceil - 1} < N < 2^{\lceil \log_2 N \rceil}$
  Now: $-2^{NH+B-1+\lceil \log_2 N \rceil} < -2^{NH+B-1}N < y \le 2^{NH+B-1}N < 2^{NH+B-1+\lceil \log_2 N \rceil}$ (we only use $N < 2^{\lceil \log_2 N \rceil}$
  From (γ), we see that we need $NH + B - 1 + \lceil \log_2 N \rceil + 1 = NH + B + \lceil \log_2 N \rceil$ bits.

  When N is a power of 2, we have that: $\log_2 N = \lceil \log_2(N+1) \rceil - 1$
  And if N is not a power of 2, we have that: $\lceil \log_2 N \rceil = \lceil \log_2(N+1) \rceil$
  From (i), and (ii), we conclude that: $Li\_FIR = NH + B + \lceil \log_2(N+1) \rceil$ (maximum # of bits for the filter).

- The case complex input, complex coefficients require also one extra register level (because of the adder/subtractor). Then: $REG\_LEVELSi = \lceil \log_2(sizeI) \rceil + \lceil \log_2(M/L) \rceil + 3$ cycles.

- For the complex FIR IP, we have added a valid output signal 'v'. We also added this to the real FIR IP.

- We also modified the FIR DA IP so that it allows anti-symmetric coefficients (SYMMETRY=AYES). These coefficients are very common when dealing with complex filters.
  This requires that we modify the adders in the symmetric mode by subtractors:
  Example: $y = h[0]x[0] + h[1]x[1] + h[2]x[2] + h[3]x[3] = h(0)(x(0) - x(3)) + h(1)(x(1) - x(3))$

N even: $N = 2M$. There are M summations:	N odd: $N = 2M-1$. There are $M-1$ summations:
$s[0] \leftarrow x[0] - x[N-1]$ $s[1] \leftarrow x[1] - x[N-2]$ $\vdots$ $s[i] \leftarrow x[i] - x[N-(i+1)]$ $\vdots$ $s[M-1] \leftarrow x[M-1] - x[N-(M-1+1)] = x[M-1] - x[M]$	$s[0] \leftarrow x[0] - x[N-1]$ $s[1] \leftarrow x[1] - x[N-2]$ $\vdots$ $s[i] \leftarrow x[i] - x[N-(i+1)]$ $\vdots$ $s[M-2] \leftarrow x[M-2] - x[N-(M-2+1)] = x[M-2] - x[M]$ $s[M-1] \leftarrow x[M-1]$

This requires the left part of the adders to be the part with the minus sign.

- The complex IP is called 'complex_FIRDA'. Figure 3 shows the IP with the mode cIcH: complex input, complex coefficients. 'sclr' clears the input register chain of each 1D FIR core. 'rst' clears the shift register for 'v' (to protect it after PR).

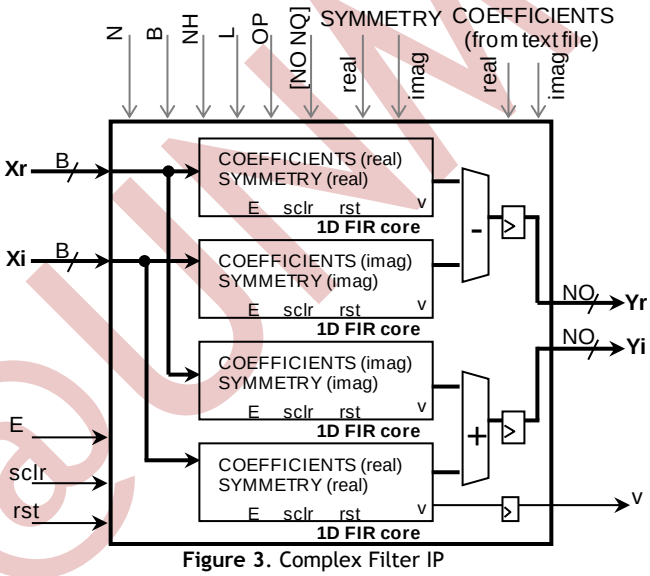


Figure 3. Complex Filter IP

PLB INTERFACE (plb_ip_cIcH_v1)

Fig. 4 shows the PLB interface for the complex filter IP. We support 3 I/O cases: B=NO=8, B=NO=16, and B=8,NO=16. This interface has been tested ONLY for the case cIcH: complex input, complex coefficients. But it will also work for the case cIrH: complex input, real coefficients (we use REG_LEVELS instead of REG_LEVELSi in this cIrH mode). For the case rIcH: real input, complex coefficients, the Input Interface will require a slight modification.

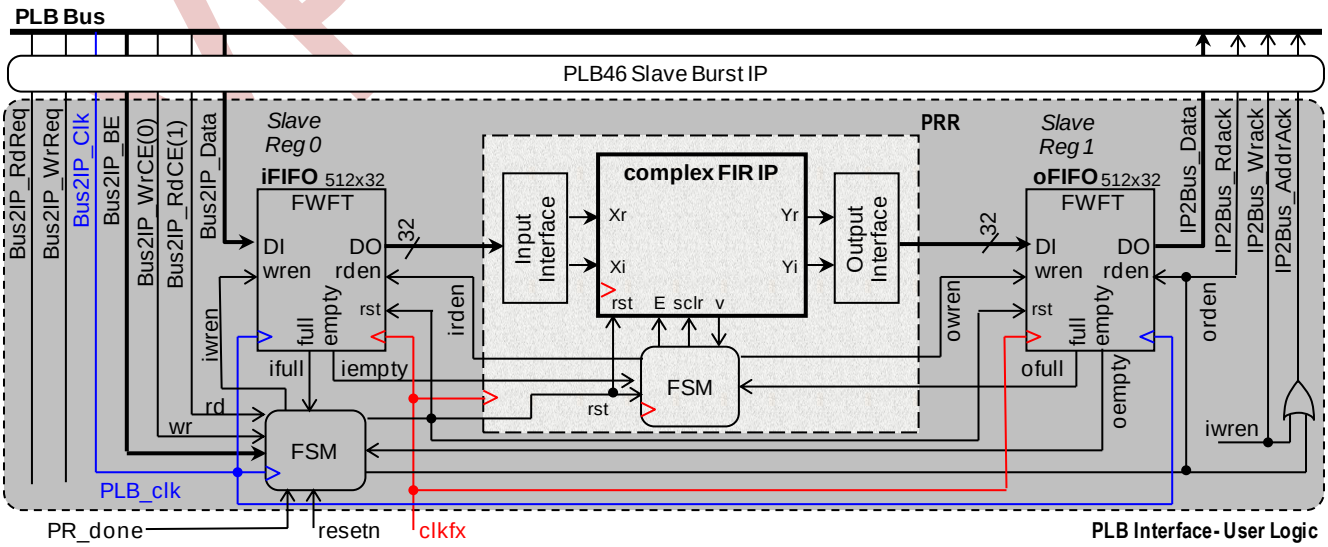


Figure 4. PLB interface for complex filter IP

FIFO18: Virtex-6 FIFO primitive. iFIFO and oFIFO have different clocks (dual clock FIFO). The following parameters need to be set: DO_REG=1, EN_SYN=FALSE → FWFT mode (no output register).
Primitives: `FIFO18E1` → 18Kb FIFO (512x36, `PLBW=32`) `FIFO36E1` → 36Kb FIFO (512x72, `PLBW=64`)
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. 5 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)

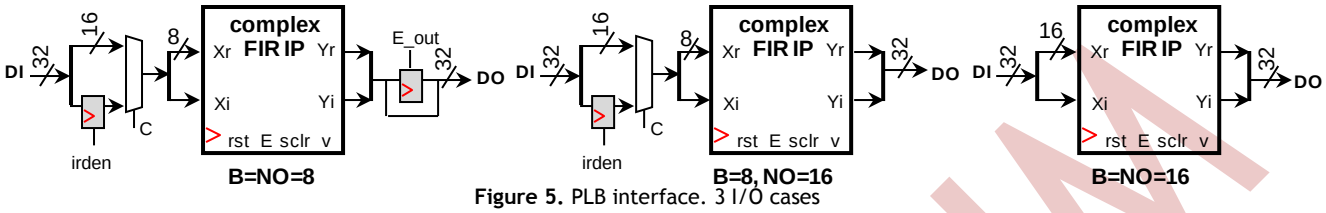


Figure 5. PLB interface. 3 I/O cases

`rst`: resets the register 'v'. `sclr`: It ONLY clears the input register chain.
* ML605: the external reset is high-level, so 'resetn' is created by inverting the external reset (this is done in the `user_logic.vhd` file)

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 6 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).

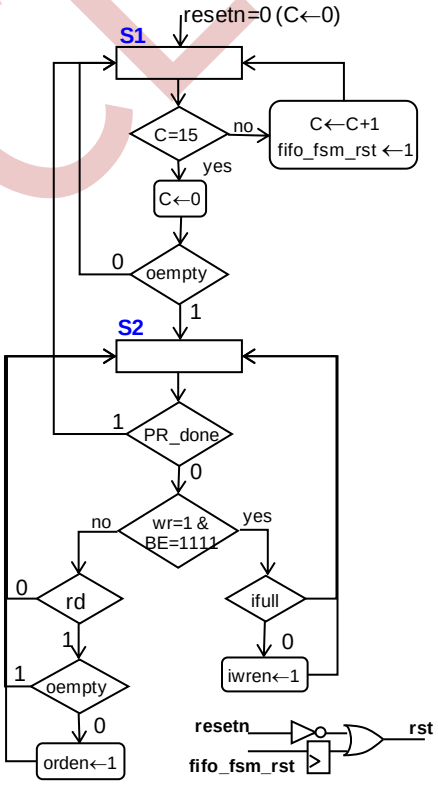


Figure 6. FSM at PLB_clk. 'C' represents a counter.

FSM @ CLKFX

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

$NWI = \frac{32}{2B}$ → # of input complex pixels contained in a 32-bit word

$NWO = \frac{32}{2NO}$ # → # of output complex pixels contained in a 32-bit word

Fig. 7 shows the complex pixel representation. If B=8, we can fit two complex pixels in a 32-bit word. If B=16, we can only fit 1 complex pixel.

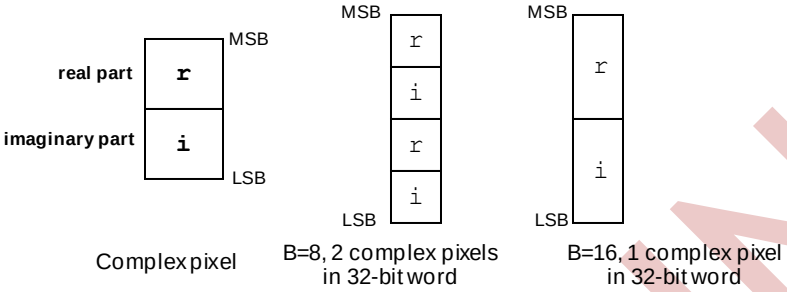


Figure 7. Complex pixels' arrangement

Fig. 8 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.

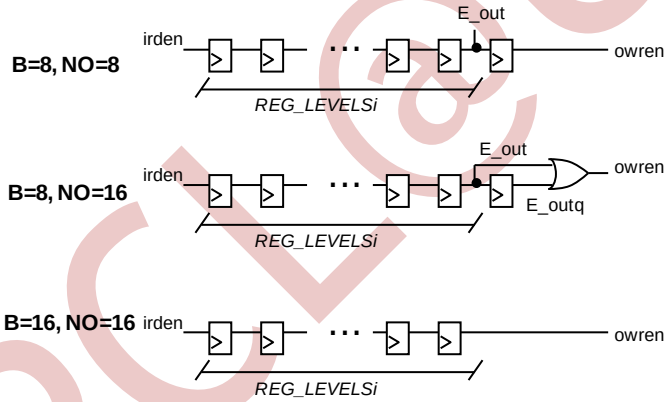


Figure 8. Generation of 'owren' for the 3 I/O cases. Note that this only works for the modes STREAM and FILT

Fig. 9 shows the timing diagram of the aforementioned cases (only STREAM and FILT) for the state machines (to be fully described later).

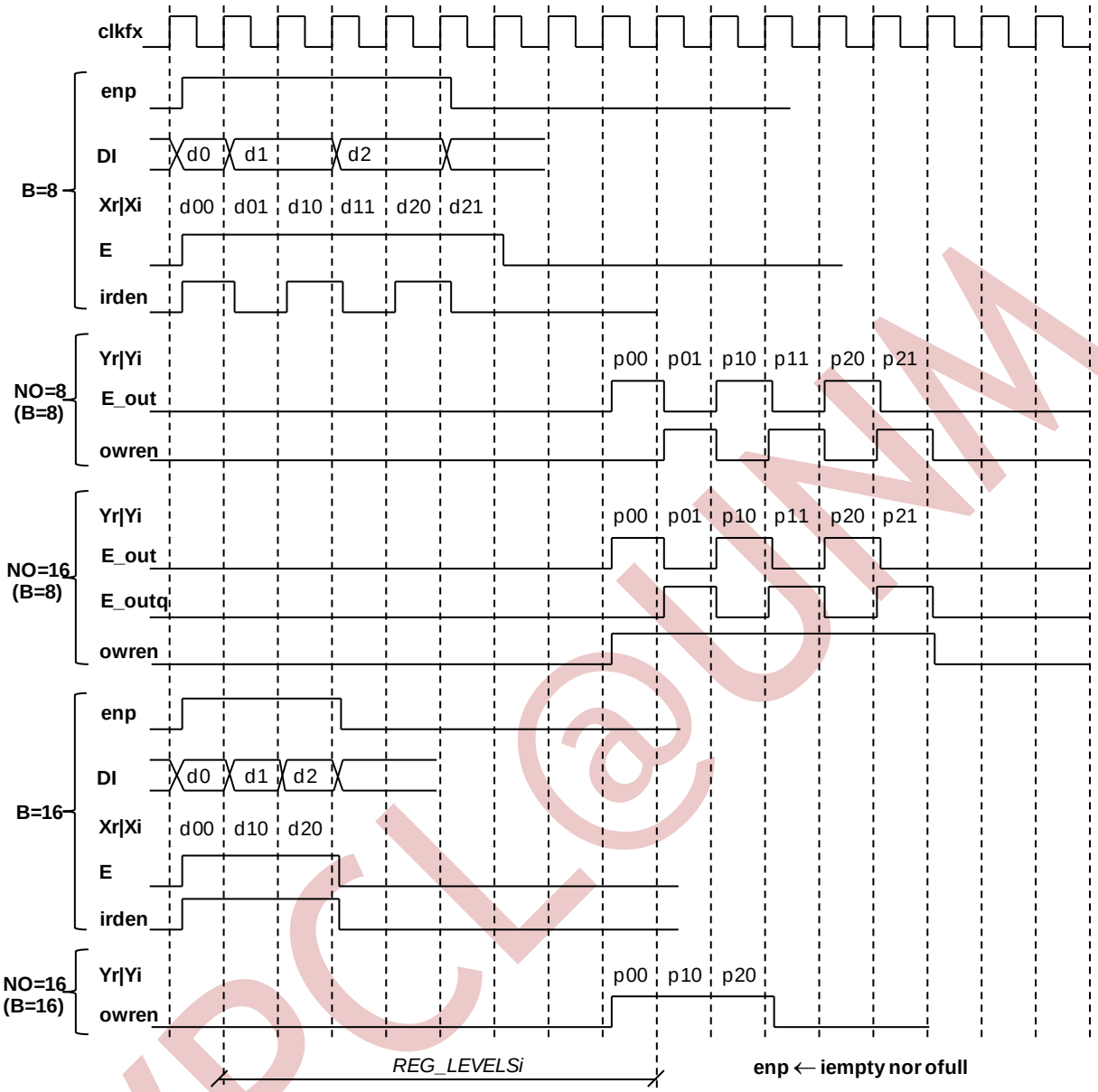


Figure 9. Timing diagram for modes STREAM and FILT. The generation or 'irden' is carried out in the State machines (to be explained later)

FSM for the mode STREAM

Fig. 10 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.

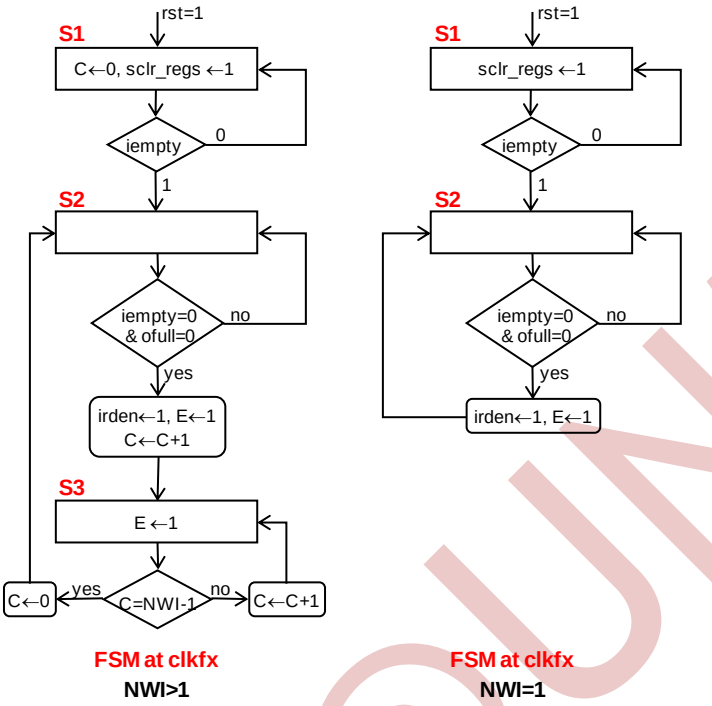


Figure 10. FSM for the STREAM mode.

FSMs for the mode IMG

Fig. 11 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*.

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.

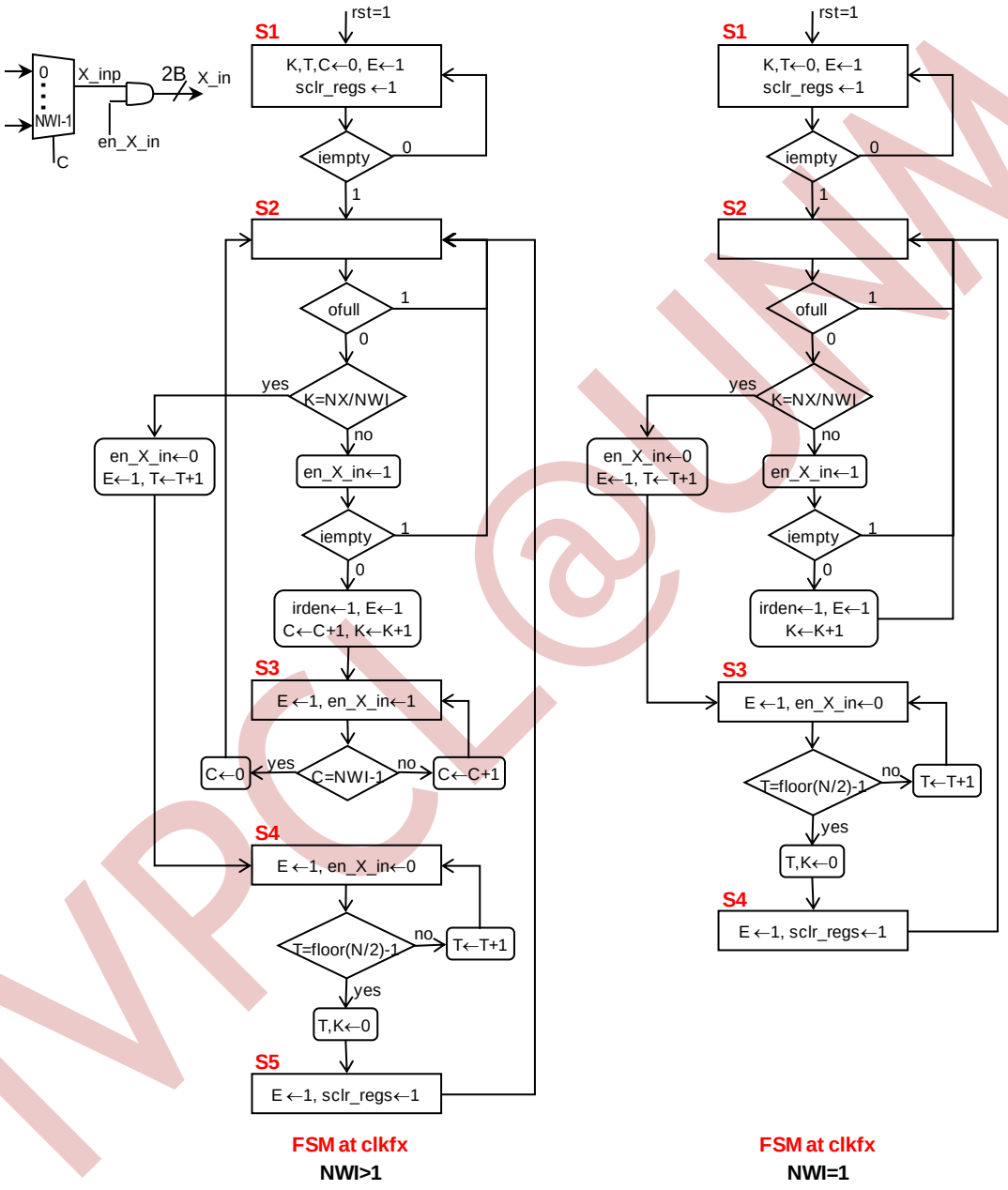


Figure 11. FSM for the IMG mode (input side)

Fig. 12 shows the FSM for the output side. '`owren`' is controlled by an FSM. The '`v`' output is employed. If *NWO=1*, then `E_out` does not exist.
The case NWO=4 does not exist for our complex filters, it is in Fig. 12 just to illustrate the idea (this can be used to improve the real filter PLB interface!)

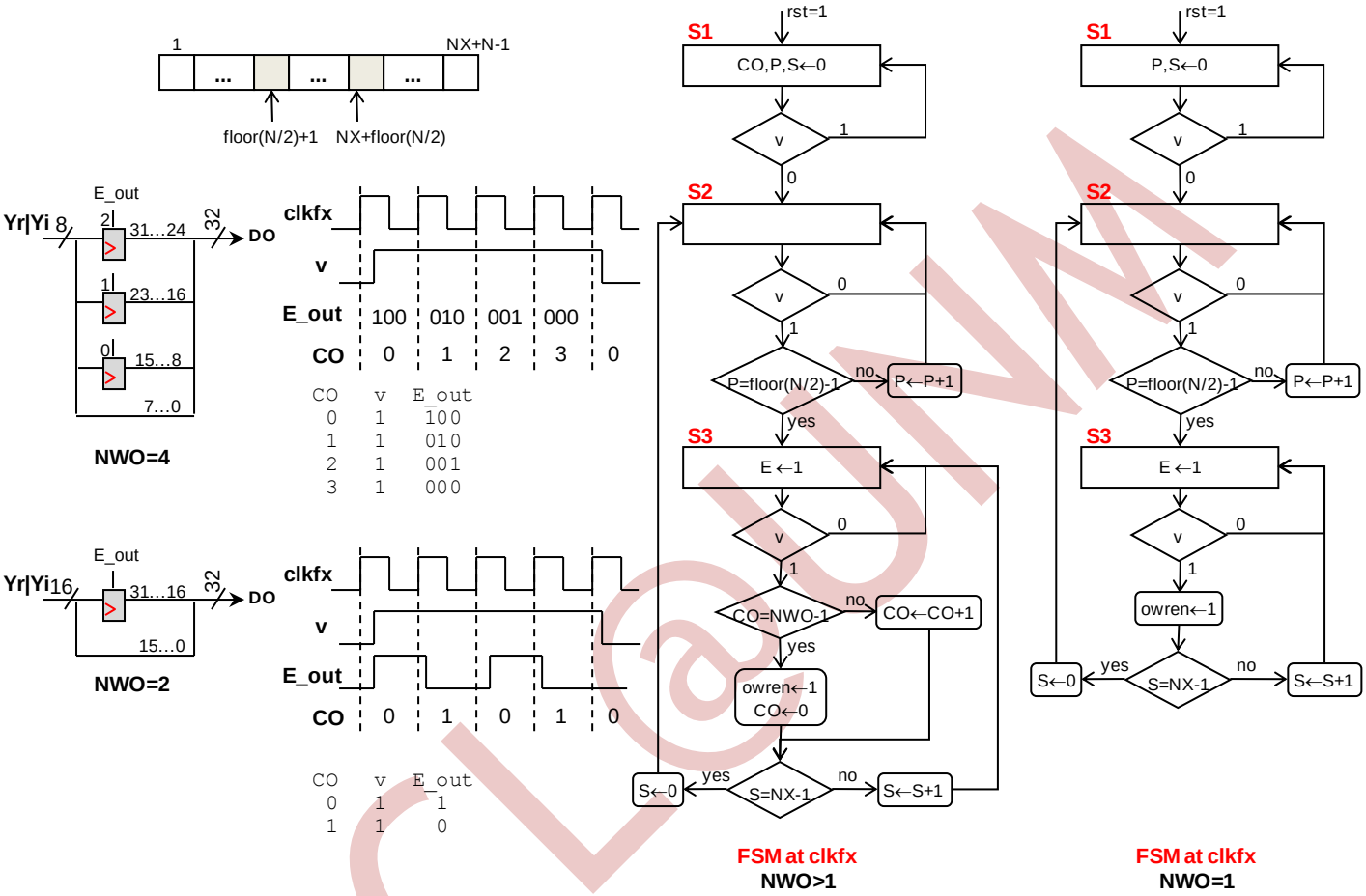


Figure 12. FSM for the IMG mode (output side)

FSM for the mode FILT

Fig. 13 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*.
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.

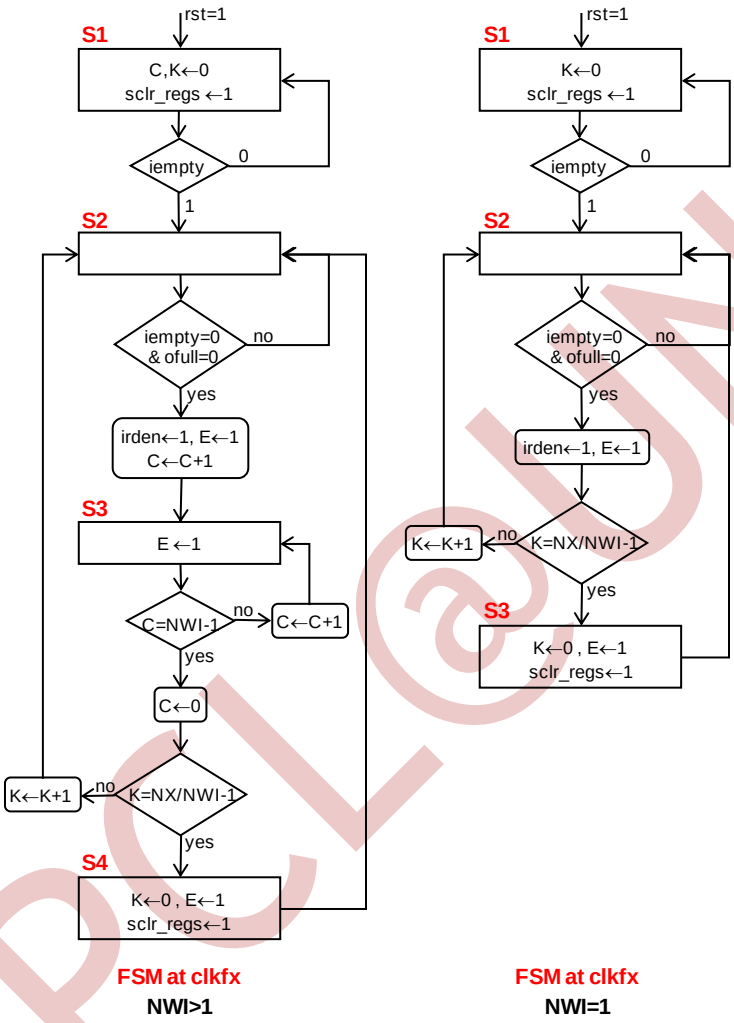


Figure 13. FSM for the FILT mode

FSMs for the mode CONV

Fig. 14 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*.

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.

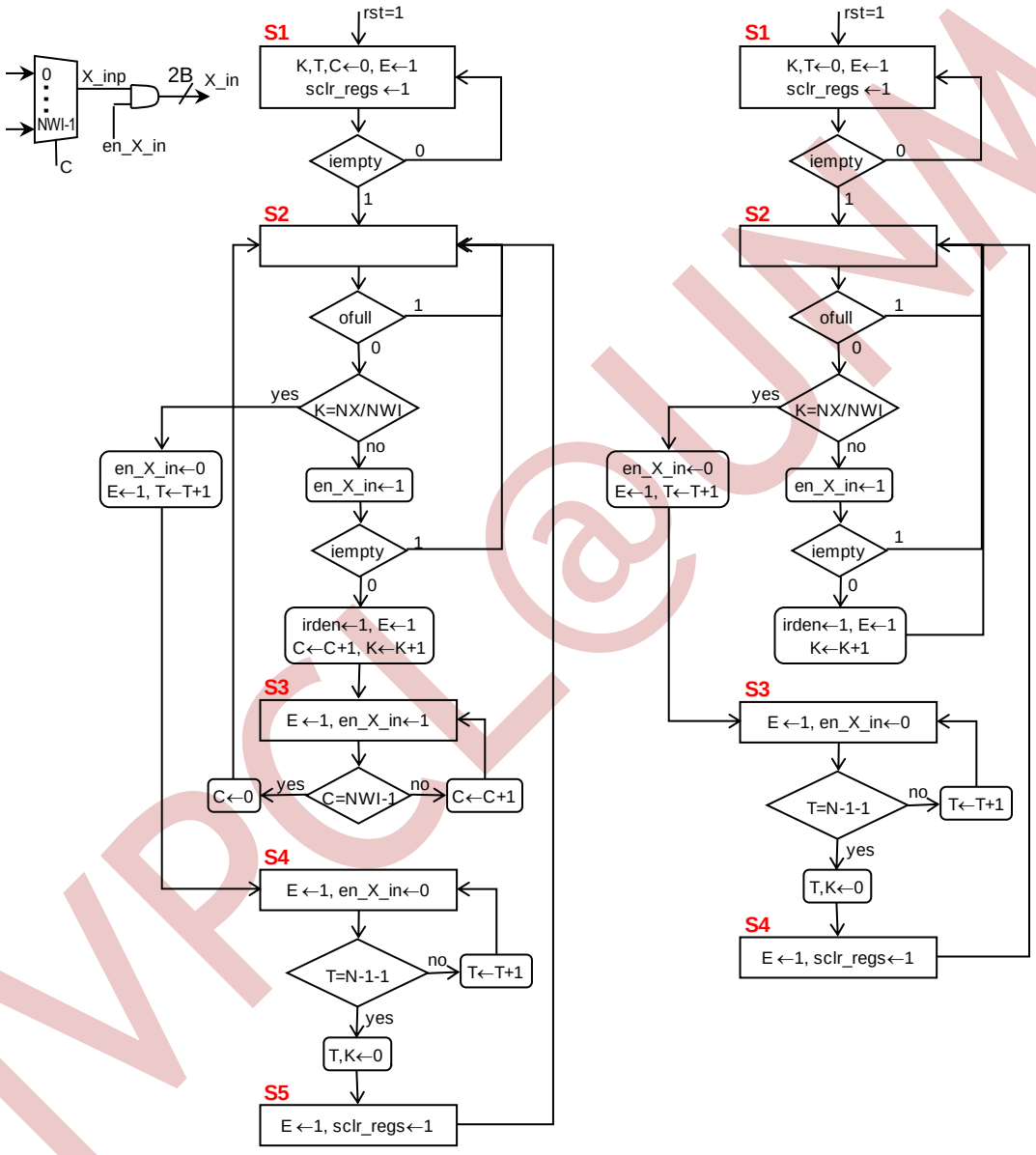
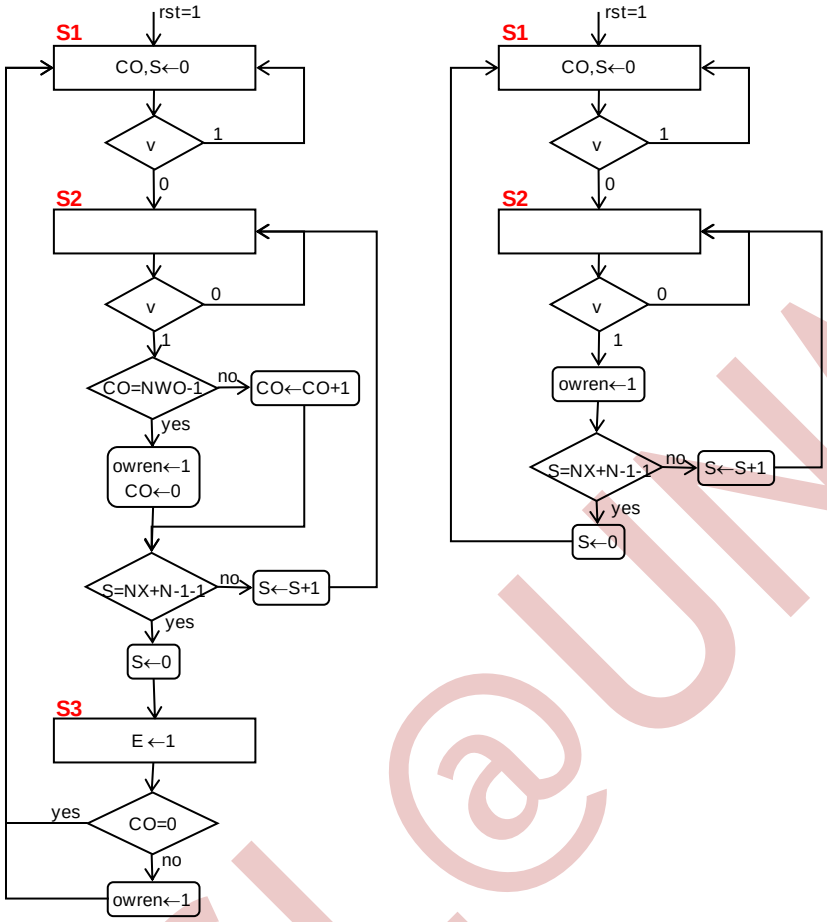


Figure 14. FSM for the CONV mode (input side)

Fig. 15 shows the FSM for the output side. '`owren`' is controlled by an FSM. The '`v`' output is employed. If *NWO=1*, then `E_out` does not exist.



FSM at clkfx
NWO>1

FSM at clkfx
NWO=1

Figure 15. FSM for the CONV mode (output side)

Figure 16 shows the Complex Filter IP (complex_FIR_DA) for the cases cIrH and rIcH.

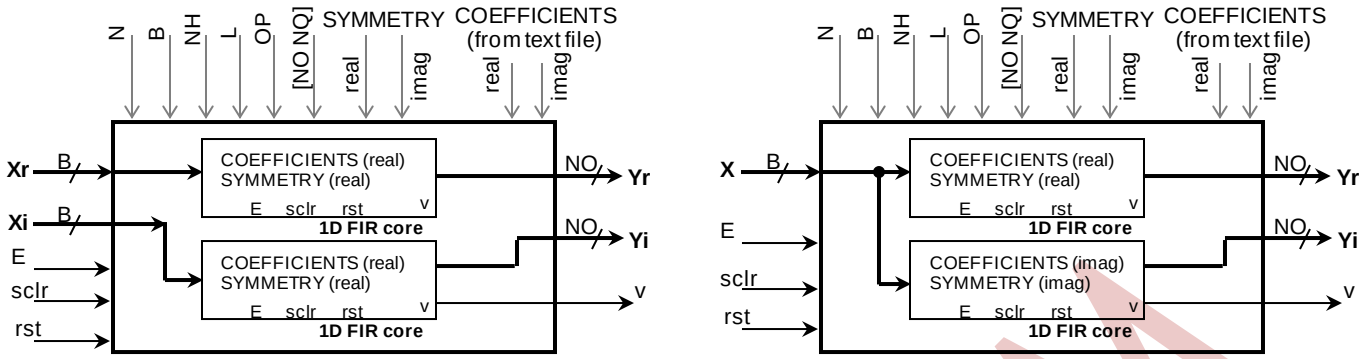


Figure 16. Complex filter IP. Complex input, real coefficients (right). Real input, complex coefficients (left)

PLB INTERFACE (plb_ip_cIrH_v1)

This interface is exactly the same as 'plb_ip_cIcH'. The only difference is the use of the complex_FIR_DA IP in the 'cIrH' mode (complex input, real coefficients).

We are using the complex_FIR_DA IP in the 'cIrH' mode (complex input, real coefficients) with an I/O delay of $REG\_LEVELS = \lceil log_2(sizeI) \rceil + \lceil log_2(M/L) \rceil + 2$ (same as the case of FIR_DA). Also, the maximum number of bits for each output (real/imaginary) is the same as in the case of the FIR_DA (real input, real coefficients).

These changes, especially the REG_LEVELS, though minor, require that we create a separate project for this case (plb_ip_cIrH_v1)

PLB INTERFACE (plb_ip_rIcH_v1)

Fig. 17 shows the PLB interface for the complex filter IP. We support 3 I/O cases: B=NO=8, B=NO=16, and B=8,NO=16. This interface has been tested ONLY for the case rIcH: real input, complex coefficients.

We are using the complex_FIR_DA IP in the 'rIcH' mode (real input, complex coefficients) with an I/O delay of $REG\_LEVELS = \lceil log_2(sizeI) \rceil + \lceil log_2(M/L) \rceil + 2$ (same as the case of FIR_DA). Also, the maximum number of bits for each output (real/imaginary) is the same as in the case of the FIR_DA (real input, real coefficients).

The interface is only different in the Input/Output Interface (see Fig. 17) and the way 'owren' is created for the cases STREAM and FILT (see Fig. 21). The FSM @PLB_Clk does not change. The FSM @clkfx is nearly identical to the case 'plb_ip_cIcH' for all modes (STREAM, IMG, FILT, CONV). The only difference is the cases IMG and CONV, where the input 'X_in' is of size 'B' bits (unlike the case 'plb_ip_cIcH' with 'X_in' of size '2B' bits); also the cases NWI=1 do not exist. Aside from these minor differences, the rest is the same.

These differences are enough to have us create a separate project for this case (plb_ip_rIcH_v1)

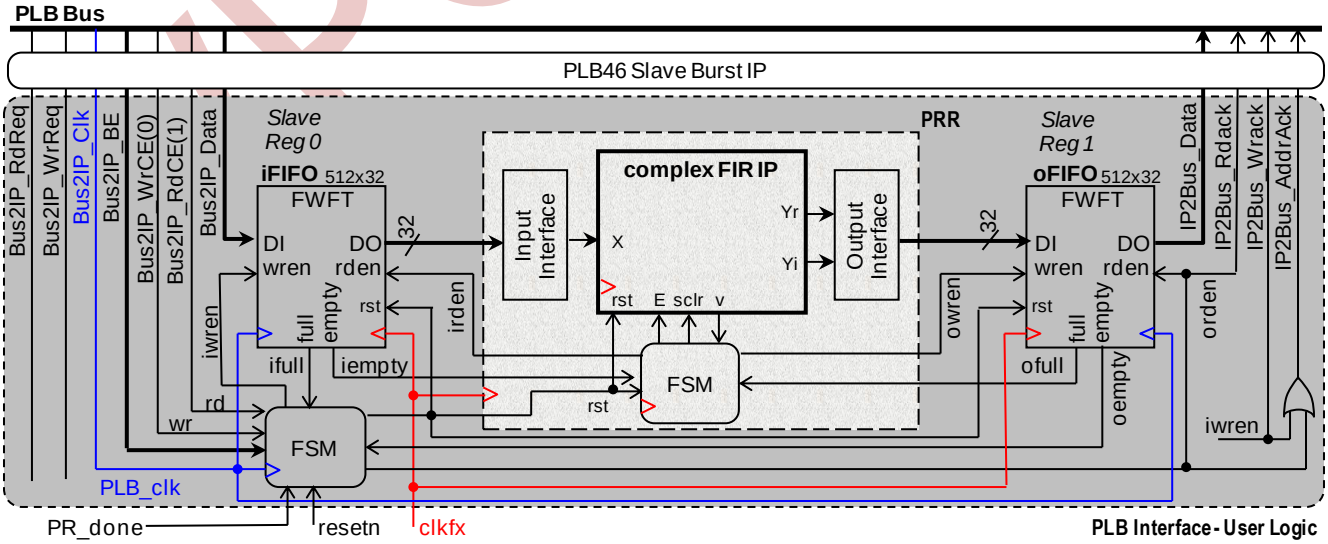


Figure 17. PLB interface for complex filter IP. Case shown: real input, complex coefficients

FIFO18: Virtex-6 FIFO primitive. iFIFO and oFIFO have different clocks (dual clock FIFO). The following parameters need to be set: DO_REG=1, EN_SYN=FALSE → FWFT mode (no output register).
Primitives: `FIFO18E1` → 18Kb FIFO (512x36, `PLBW=32`) `FIFO36E1` → 36Kb FIFO (512x72, `PLBW=64`)
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. 18 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)

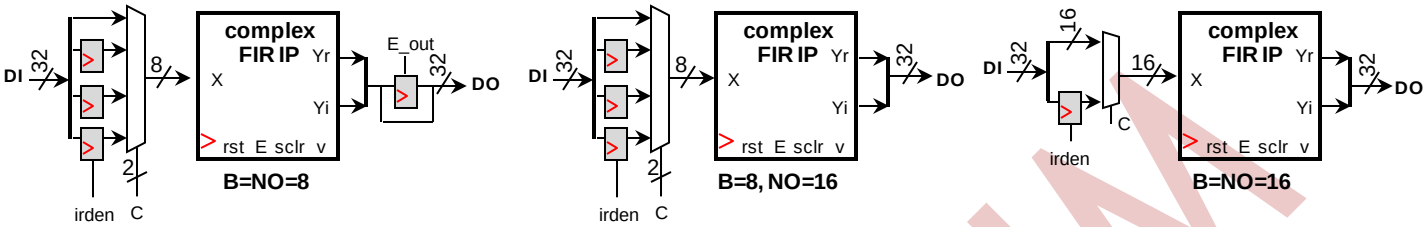


Figure 18. PLB interface. 3 I/O cases

`rst`: resets the register 'v'. `sclr`: It ONLY clears the input register chain.
* ML605: the external reset is high-level, so 'resetn' is created by inverting the external reset (this is done in the `user_logic.vhd` file)

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 19 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).

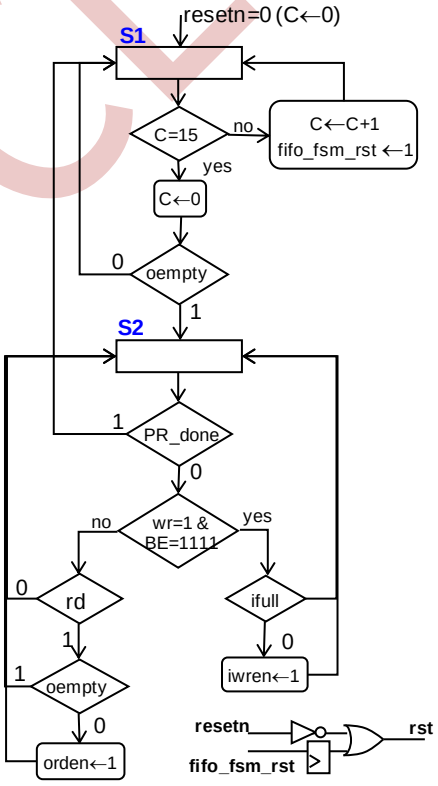


Figure 19. FSM at PLB_clk. 'C' represents a counter.

FSM @ CLKFX

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

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

$NWO = \frac{32}{2NO}$ # → # of output complex pixels contained in a 32-bit word

Fig. 20 shows the real (input) and complex (output) pixel representation. If NO=8, we can fit two complex pixels in a 32-bit word. If NO=16, we can only fit one complex pixel.

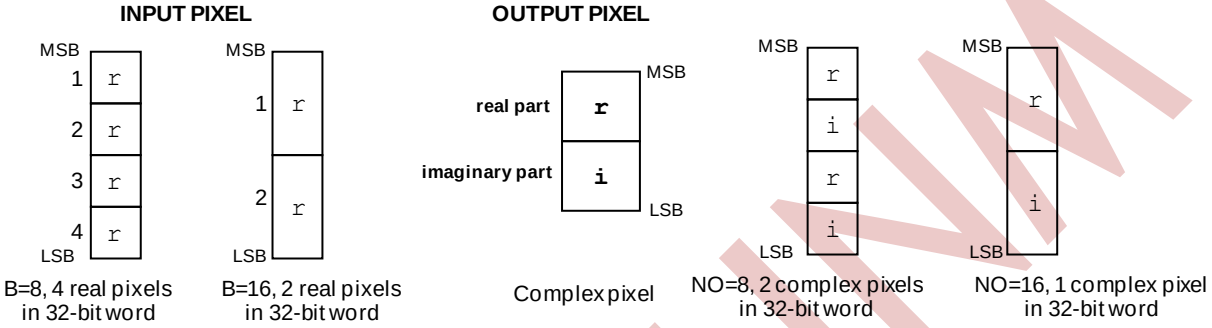


Figure 20. Complex pixels' arrangement

Fig. 21 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.

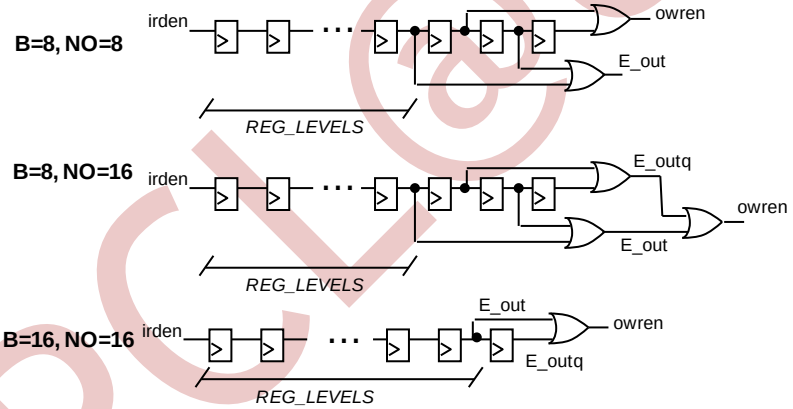


Figure 21. Generation of 'owren' for the 3 I/O cases. Note that this only works for the modes STREAM and FILT

Fig. 22 shows the timing diagram of the aforementioned cases (only STREAM and FILT) for the state machines (to be fully described later).

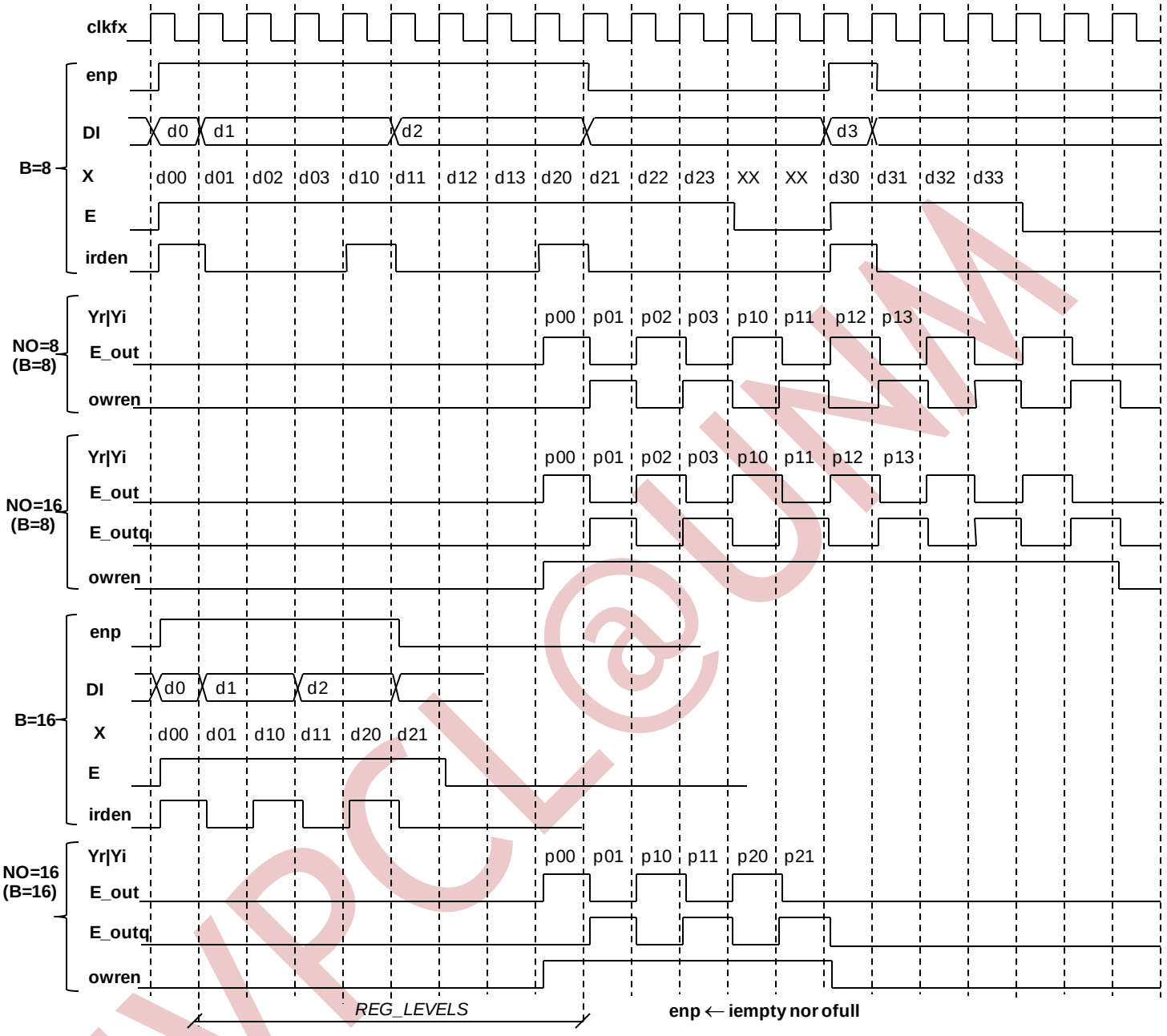


Figure 22. Timing diagram for modes STREAM and FILT. The generation or 'irden' is carried out in the State machines (to be explained later)

FSM for the mode STREAM

Fig. 23 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.

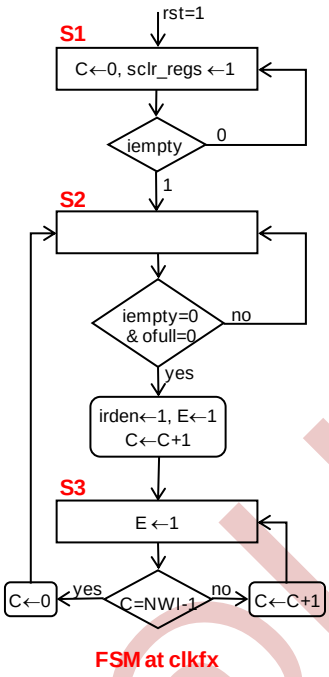


Figure 23. FSM for the STREAM mode.

FSMs for the mode IMG

Fig. 24 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*.

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.

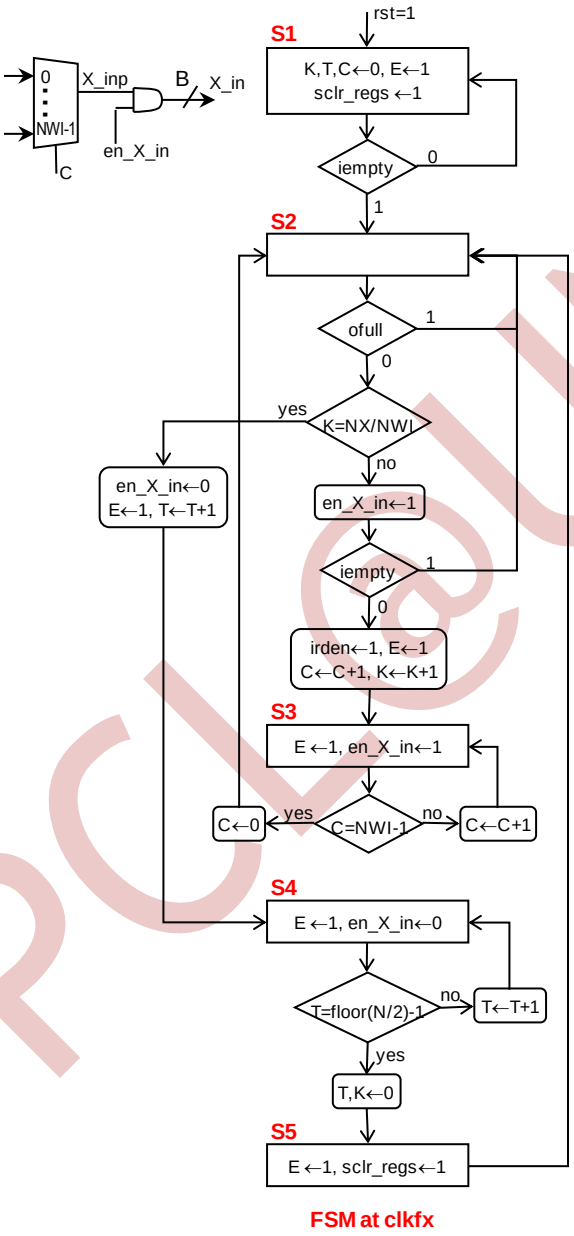


Figure 24. FSM for the IMG mode (input side)

Fig. 25 shows the FSM for the output side. '`owren`' is controlled by an FSM. The '`v`' output is employed. If *NWO=1*, then `E_out` does not exist.
The case NWO=4 does not exist in this case (rIcH), it is in the figure just to illustrate the idea.

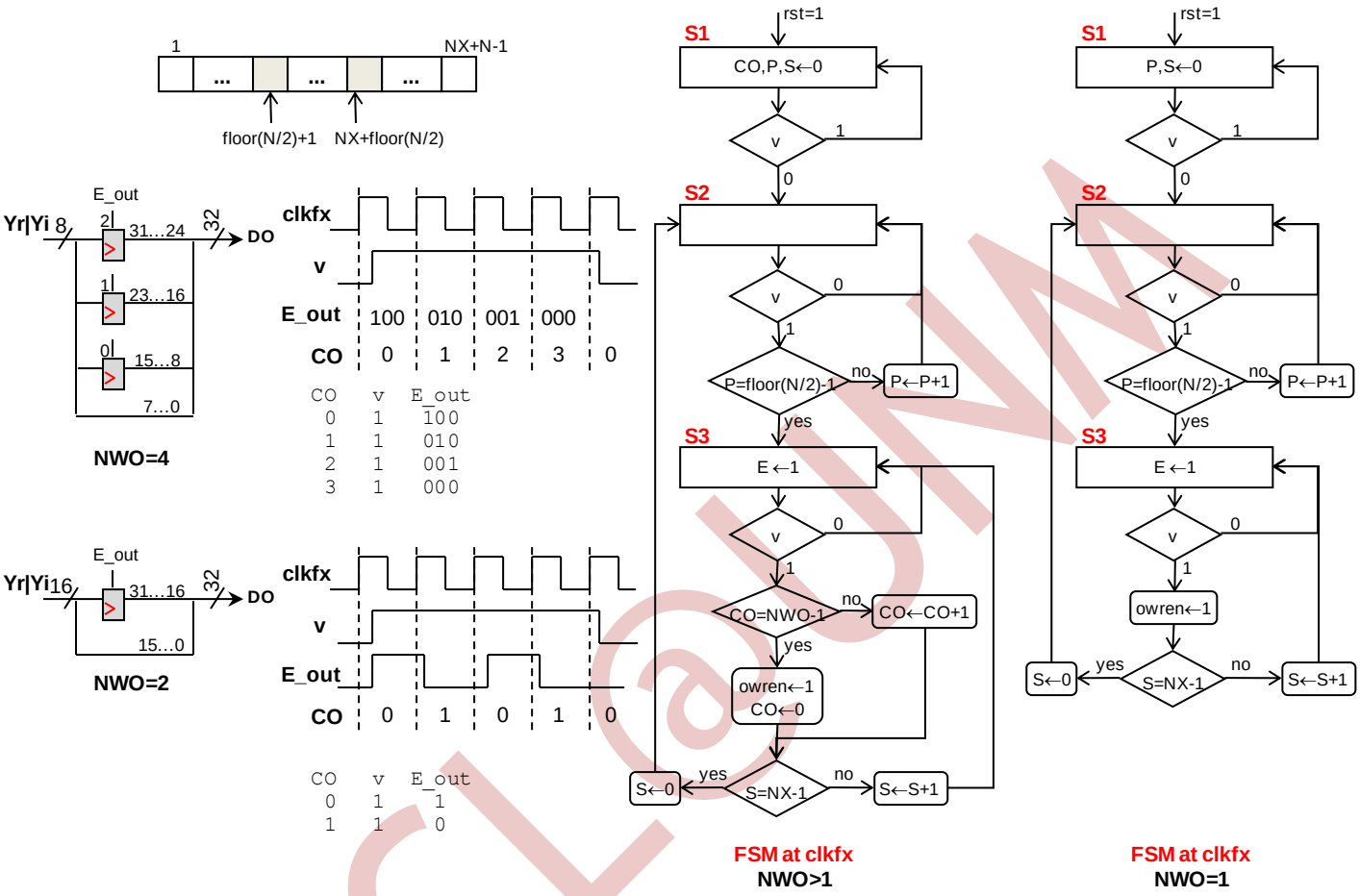


Figure 25. FSM for the IMG mode (output side)

FSM for the mode FILT

Fig. 26 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*.
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.

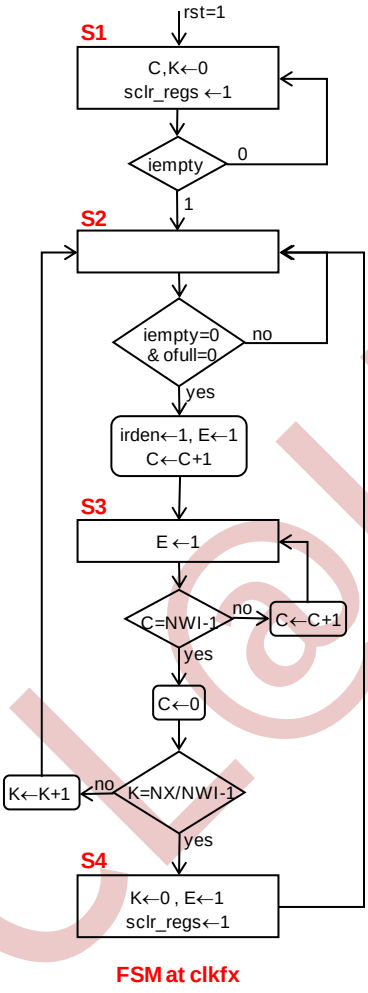


Figure 26. FSM for the FILT mode

FSMs for the mode CONV

Fig. 27 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*.

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.

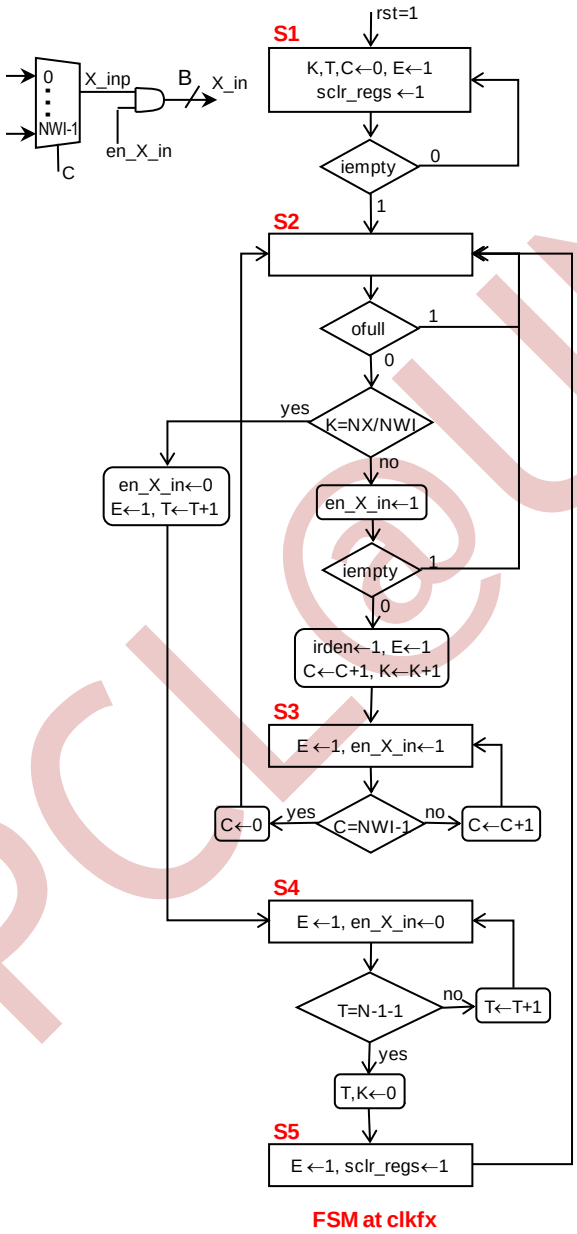


Figure 27. FSM for the CONV mode (input side)

Fig. 28 shows the FSM for the output side. '`owren`' is controlled by an FSM. The '`v`' output is employed. If *NWO=1*, then `E_out` does not exist.

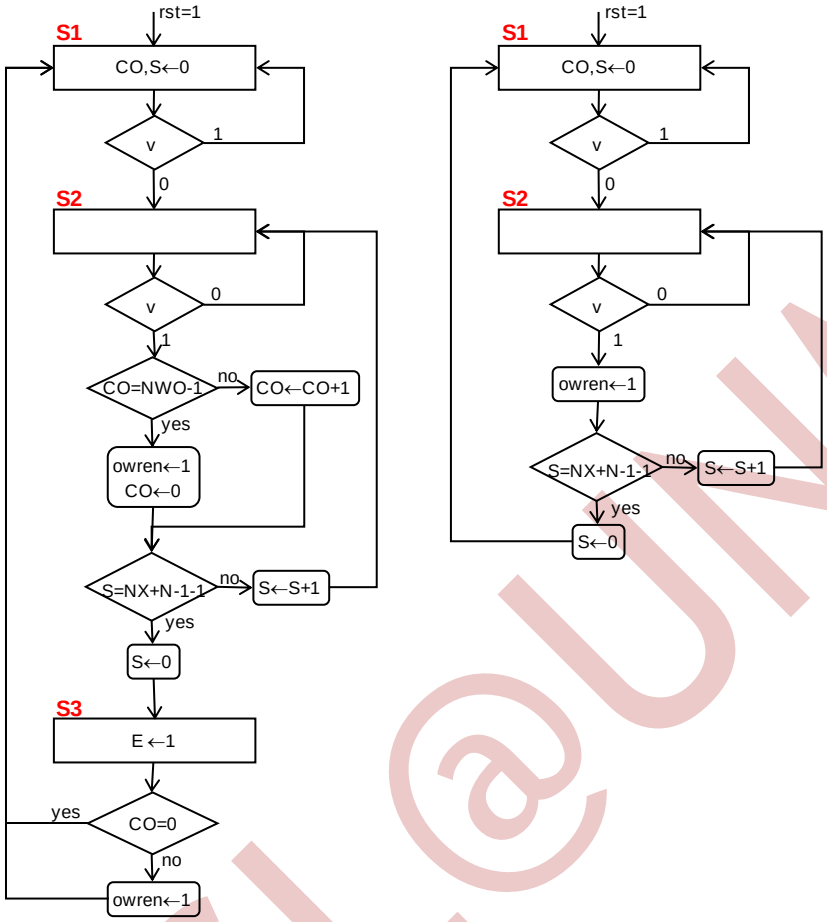


Figure 28. FSM for the CONV mode (output side)