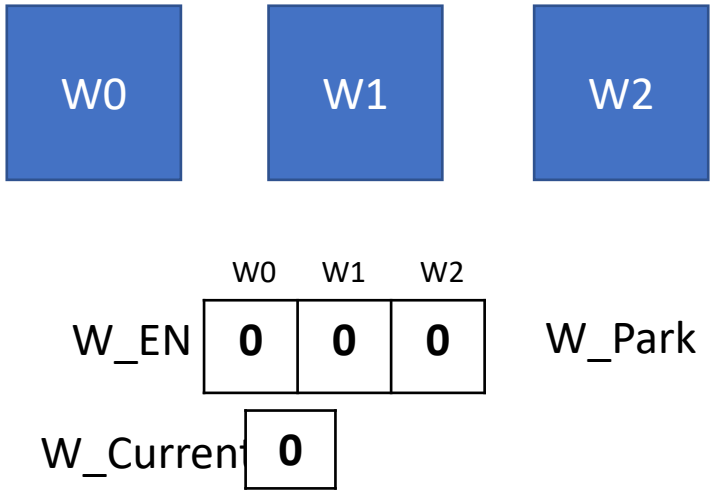
Sprzętowa implementacja splotowej filtracji sygnału video

Radosław Feiglewicz

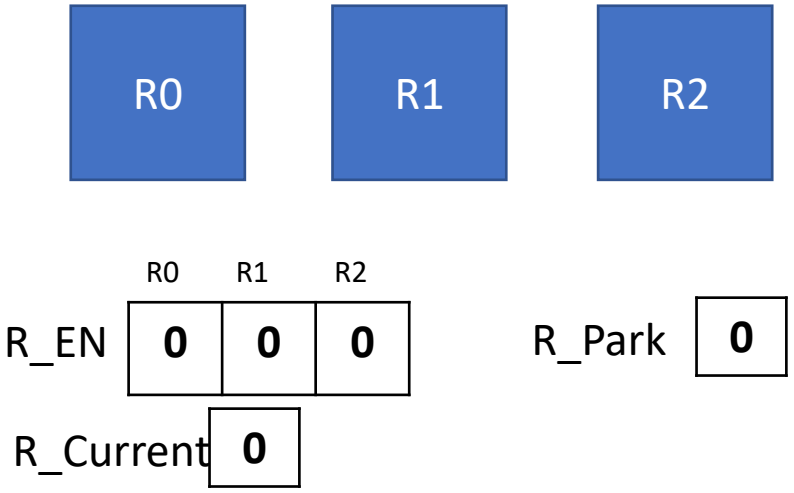
Dominik Jaworski

Algorytm synchronizacji VDMA z filtrowaną ramką

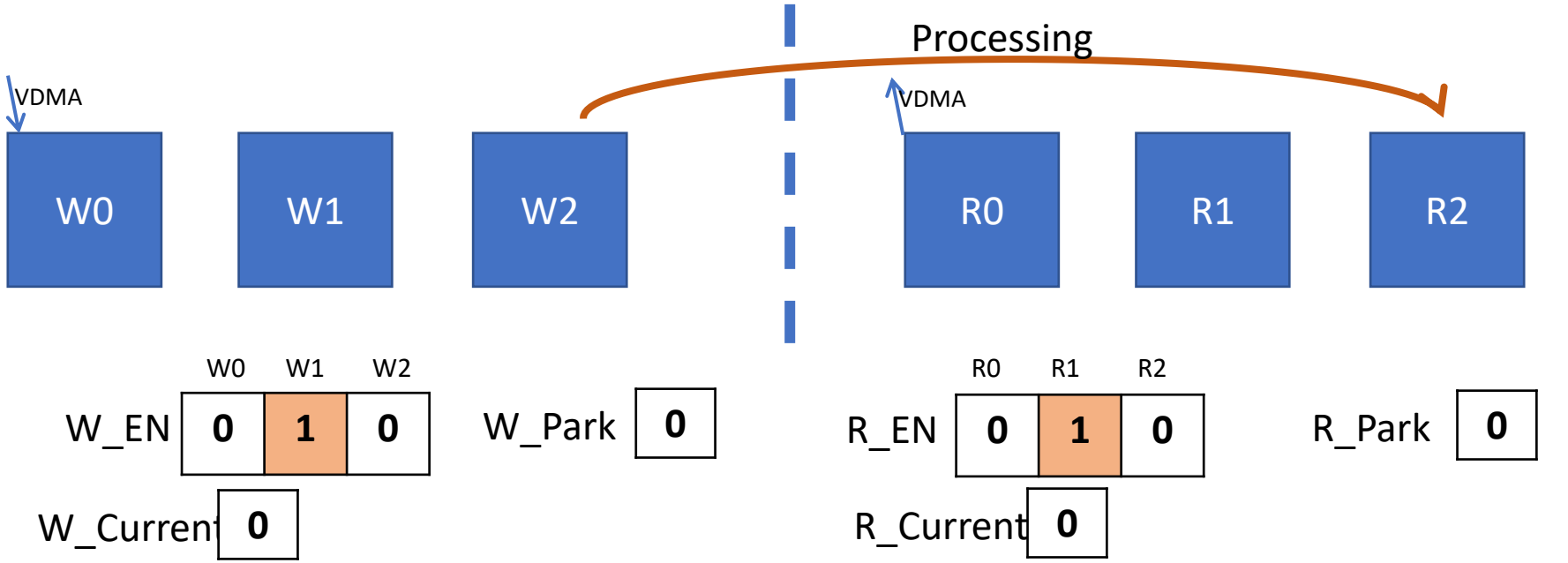
Zapis z kamery do pamięci



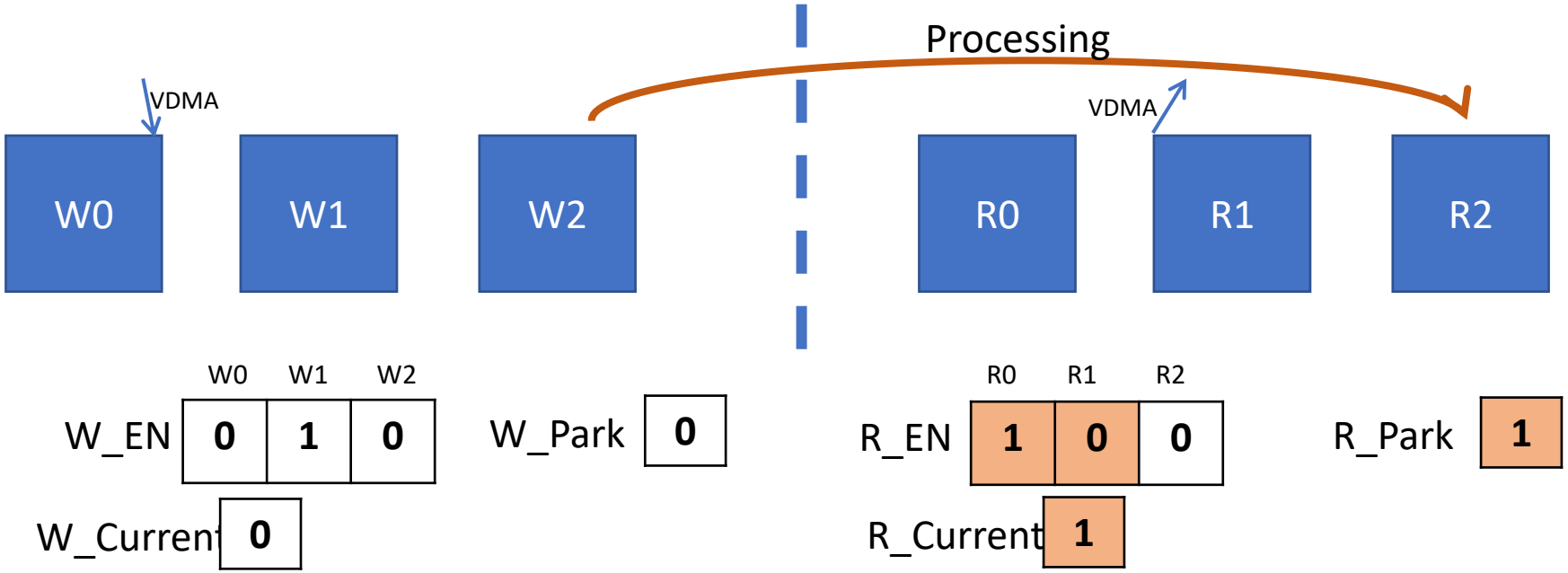
Odczyt z pamięci i przesyłanie do HDMI



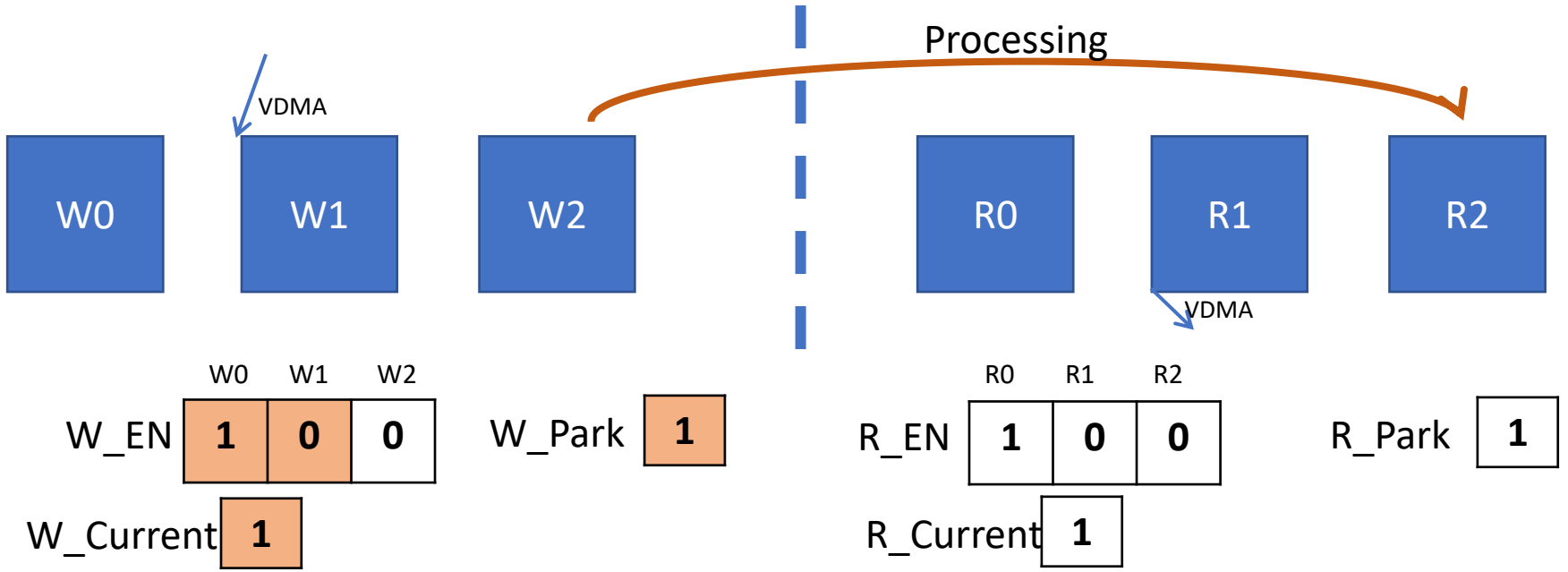
Kwadraty W0,W1,W2 oznaczają przestrzenie w pamięci DDR w których zapisywane będą kolejne klatki odczytane z kamery. Natomiast Kwadraty R0,R1,R2 oznaczają również przestrzenie w DDR ale dla danych już przetworzonych. Tablice W_EN i R_EN informują który obszar (np.W0) nie jest aktualnie wykorzystywany(w przypadku W0 czy jest wykorzystywany do zapisu z kamery lub aktualnie procesowany przez algorytm filtrujący. Rejestry R_Park i W_Park informują czy moduł VDMA jest w trybie parkującym tzn. po skończeniu przetwarzania klatki dokonuje przetwarzania na tej samej przestrzeni pamięci co poprzednio (jeśli zapisywał do W0 to ponownie będzie zapisywał do W0). Rejestry W_Current i R_Current mówią która przestrzeń jest teraz przetwarzana(jeśli W_Current == 1 to znaczy że W1



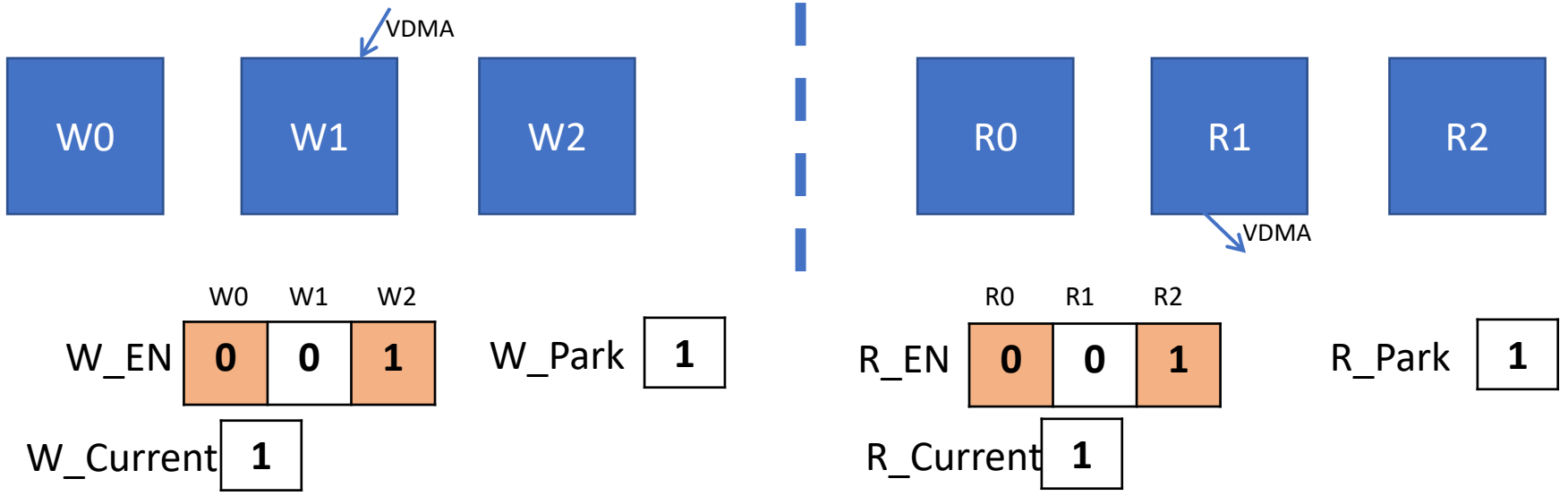
Start – Do W0 będzie zapisywana klatka natomiast z W2 dane będą przetwarzane i zapisywane do R2. Natomiast dane z R0 są odczytywane. Dlatego W_EN[1] =1 i R_EN[1] = 1 ponieważ tam nie ma danych i nie wykonywane są tam żadne operacje.



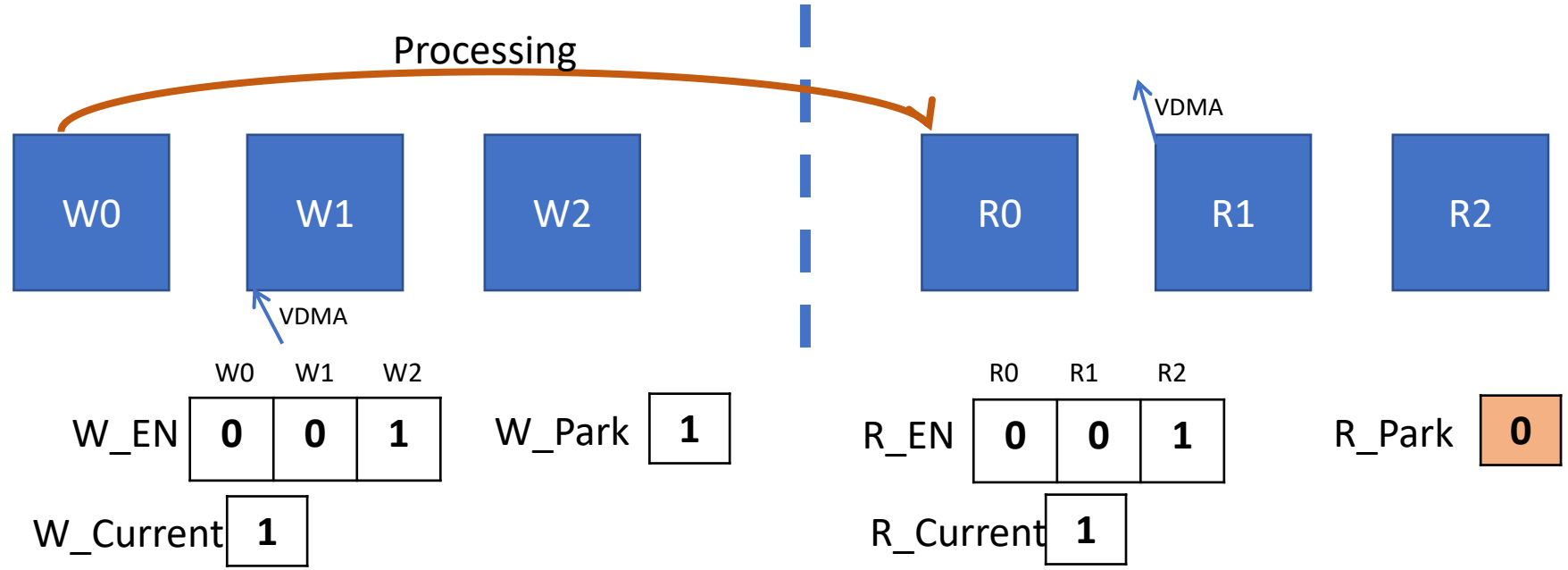
Kanał read zakończył odczytywanie ramki R0 i automatycznie przechodzi do odczytu ramki R1 (bo R_Park == 0) i wywołuje przerwanie. W callbacku sprawdzane jest czy R_EN[2] jest 0. Processing nadal trwa dlatego trzeba będzie się zatrzymać na klatce R1 do momentu skończenia processingu. Dlatego R_Park = 1. R_Current jest inkrementowany



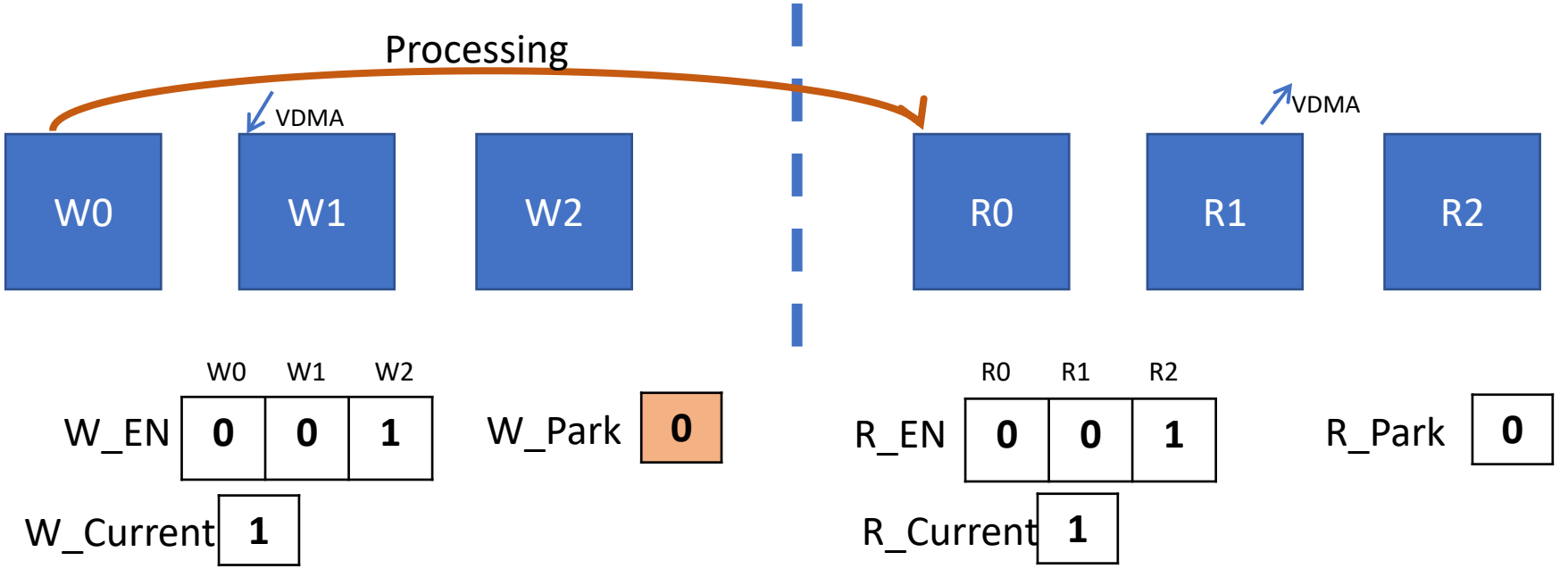
Kanał write zakończył odczytywanie ramki W0 i automatycznie przechodzi do zapisu ramki W1 (bo W_Park == 0) i wywołuje przerwanie. W callbacku sprawdzane jest czy W_EN[2] jest 0. Processing nadal trwa dlatego trzeba będzie się zatrzymać na klatce W1 do momentu skończenia processingu. Dlatego W_Park = 1. W_Current jest inkrementowany



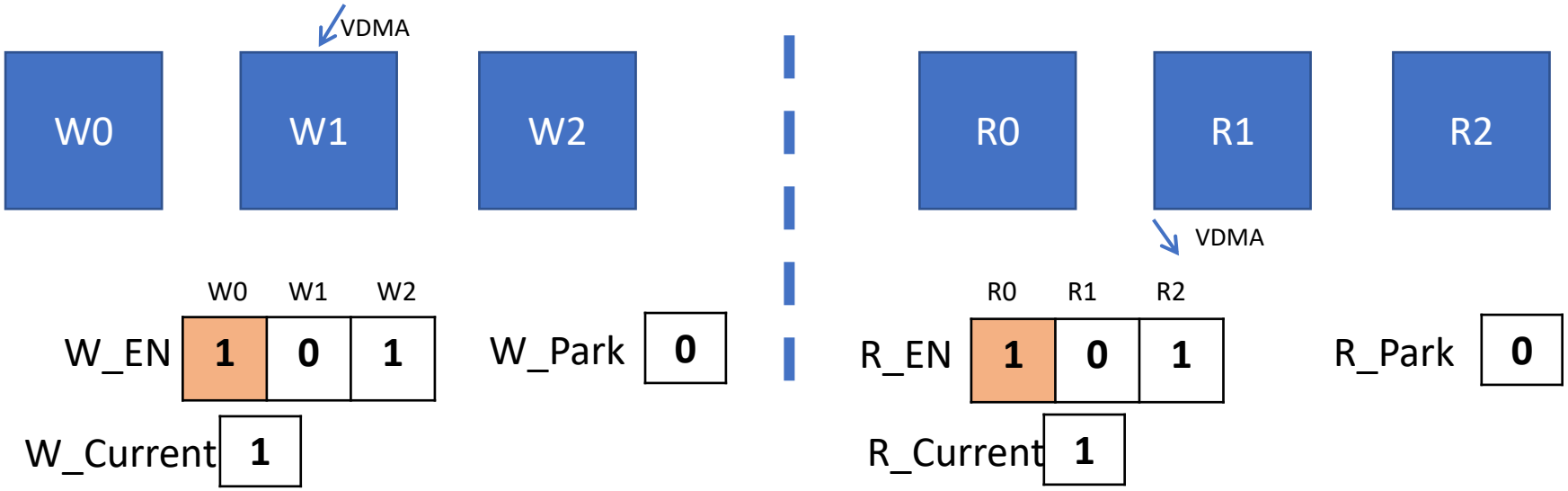
Algorytm skończył procesować W2 z R2 ustawia ich rejestry na 1 i czeka z następnym procesingiem do momentu jak kolejna przestrzeń modulo 3 jest dostępna. W_EN[0] i R_EN[0] są 1 dlatego może ich użyć i ustawia rejestry na 0



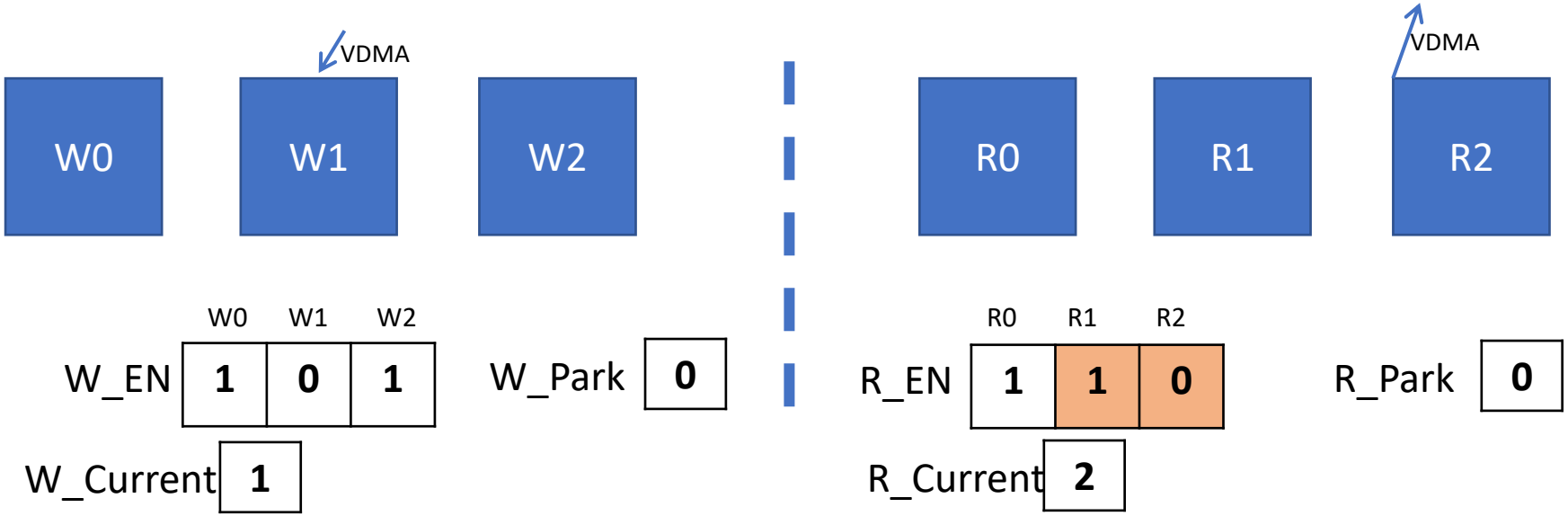
Odczyt z R1 zakończył się ale w poprzednim read-callbecku ustawiony został R_Park i odczyt będzie dalej z tej samej klatki .Ponieważ R[2] jest wolny można R_park wyłączyć.



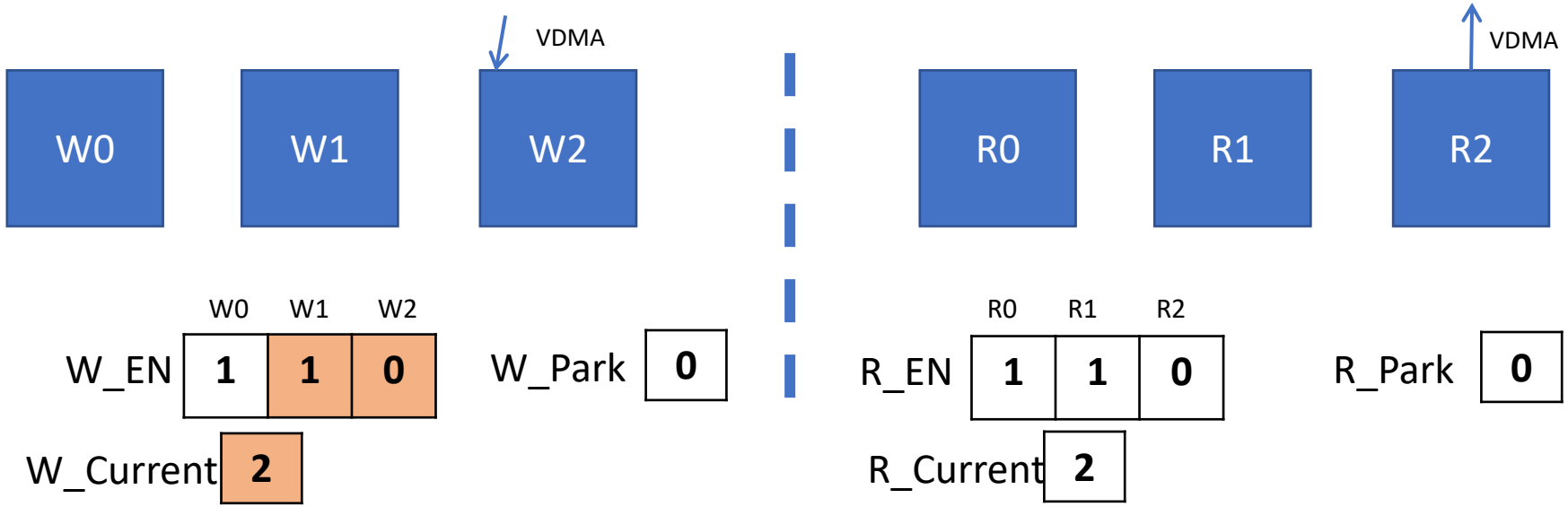
Zapis do W1 zakończył się ale w poprzednim write-callbecku ustawiony został W_Park i zapis będzie dalej do tej samej klatki.Ponieważ W[2] jest wolny można W_park wyłączyć.



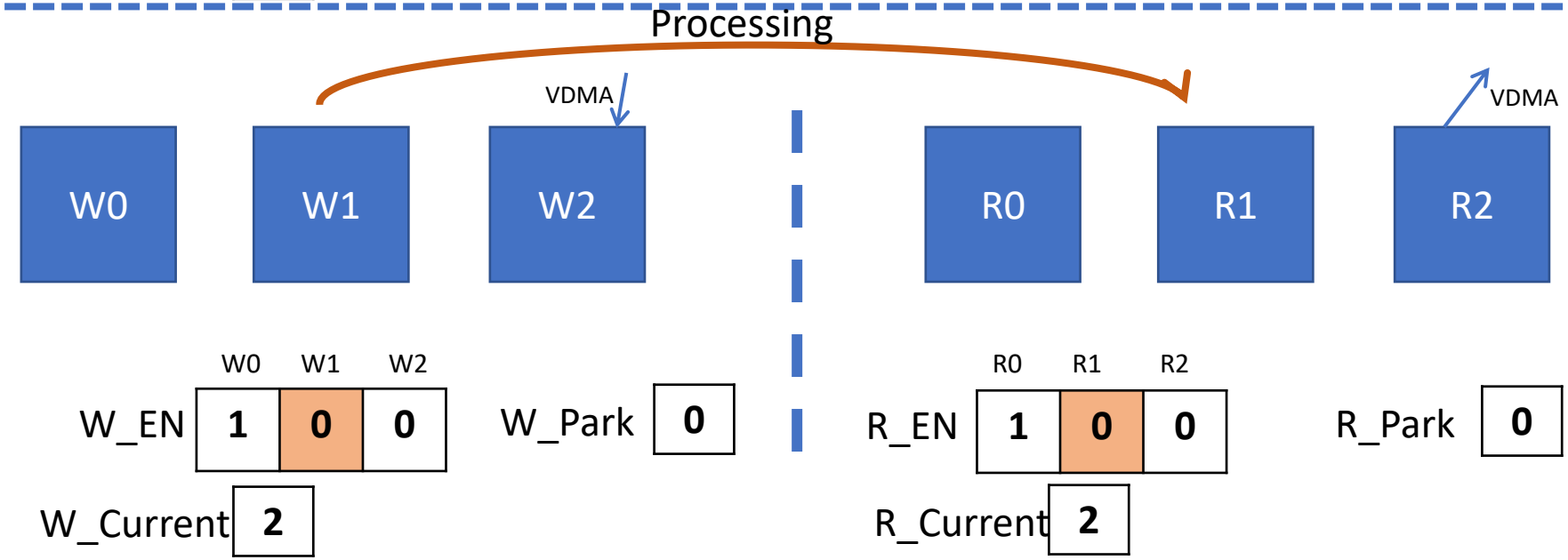
Algorytm skończył procesować W0 z R0 ustawia ich rejestry na 1 i czeka z następnym procesingiem do momentu jak kolejna przestrzeń modulo 3 jest dostępna. W_EN[1] i R_EN[1] są 0 dlatego musi czekać



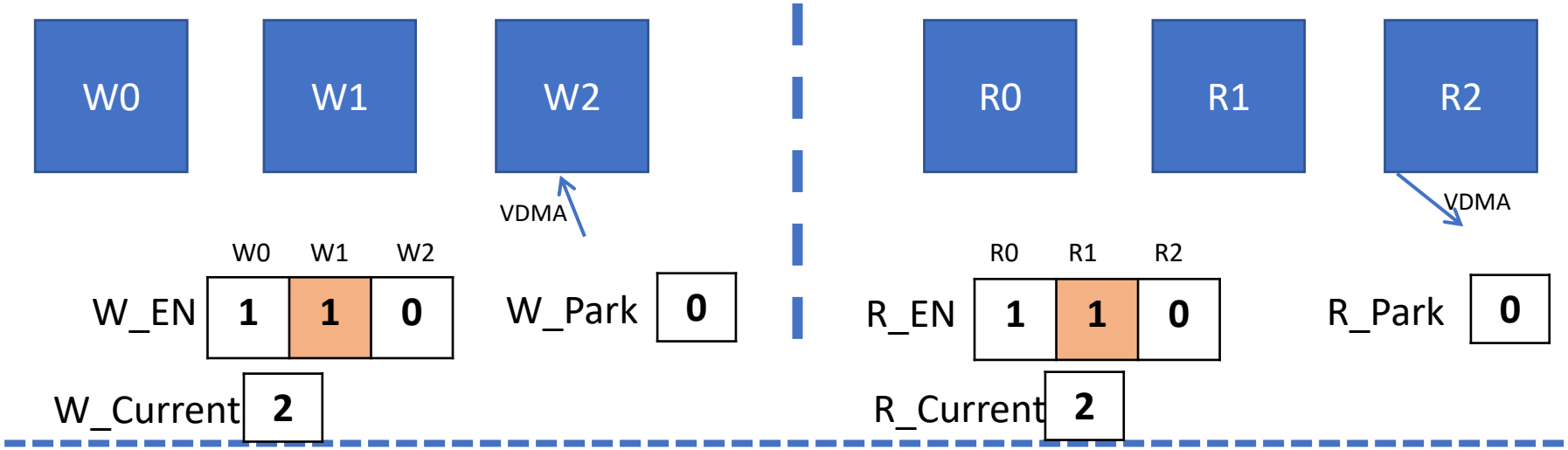
Kanał read zakończył odczytywanie ramki R1 i automatycznie przechodzi do odczytu ramki R2 (bo R_Park == 0) i wywołuje przerwanie. W callbacku sprawdzane jest czy R_EN[0] jest 0. R_EN[0] jest wolne dlatego nie parkujemy. R_Current inkrementujemy



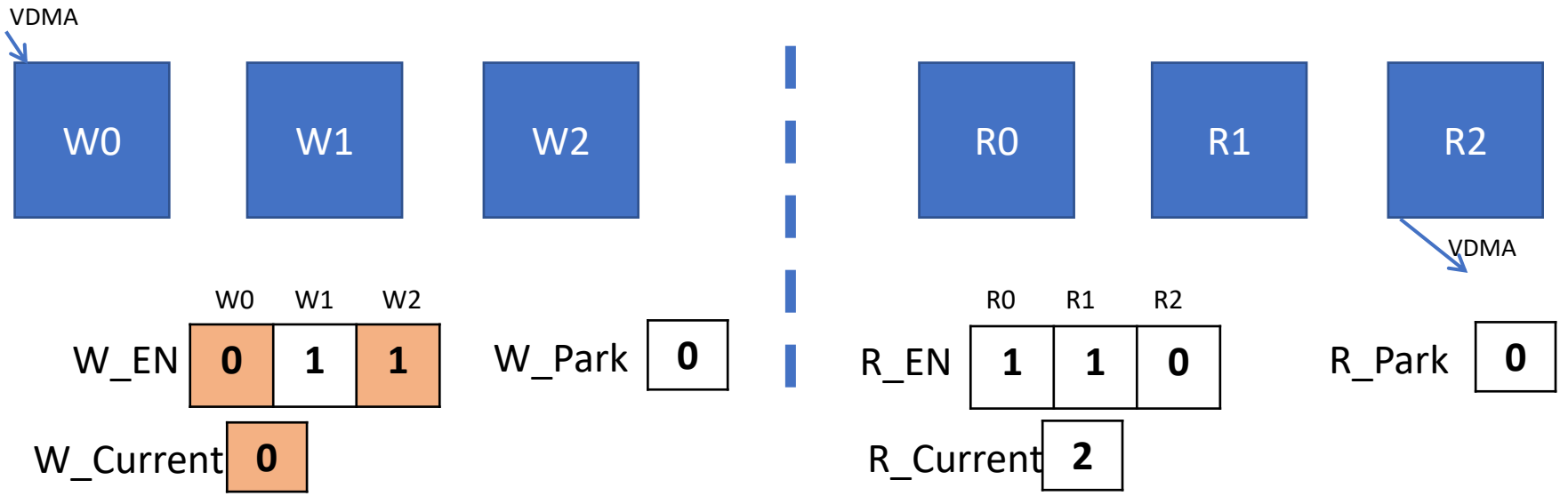
Kanał write zakończył odczytywanie ramki W1 i automatycznie przechodzi do zapisu ramki W2 (bo W_Park == 0) i wywołuje przerwanie. W callbacku sprawdzane jest czy W_EN[3] jest 0. Klatka jest wolna dlatego tylko inkrementujemy W_Current



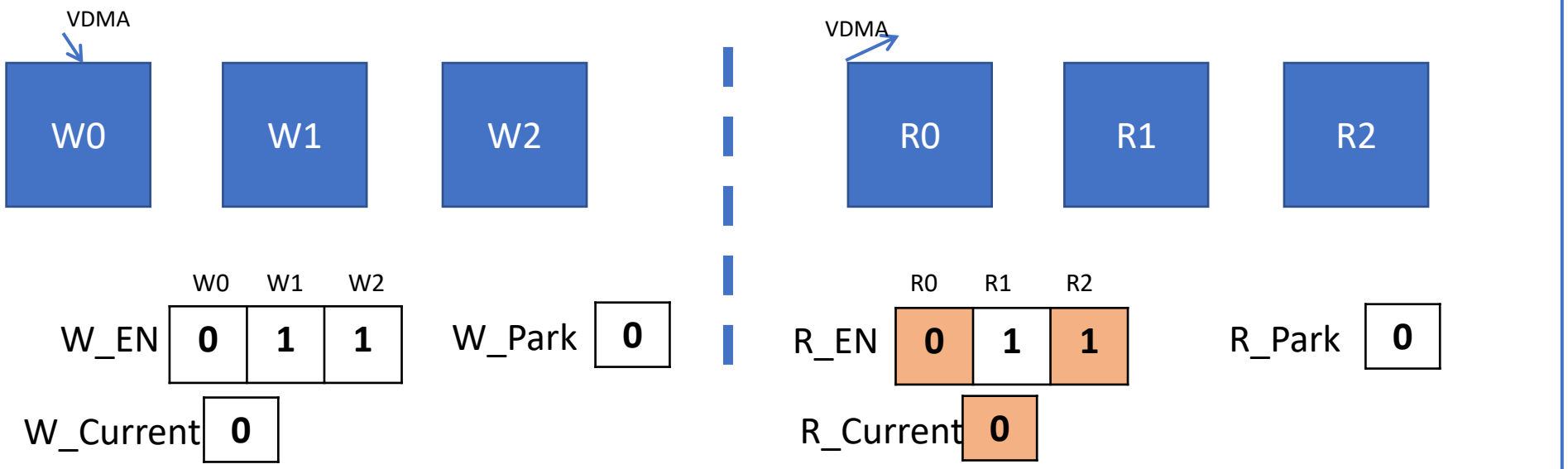
Ponieważ W1 i R1 są wolne następuje processing.



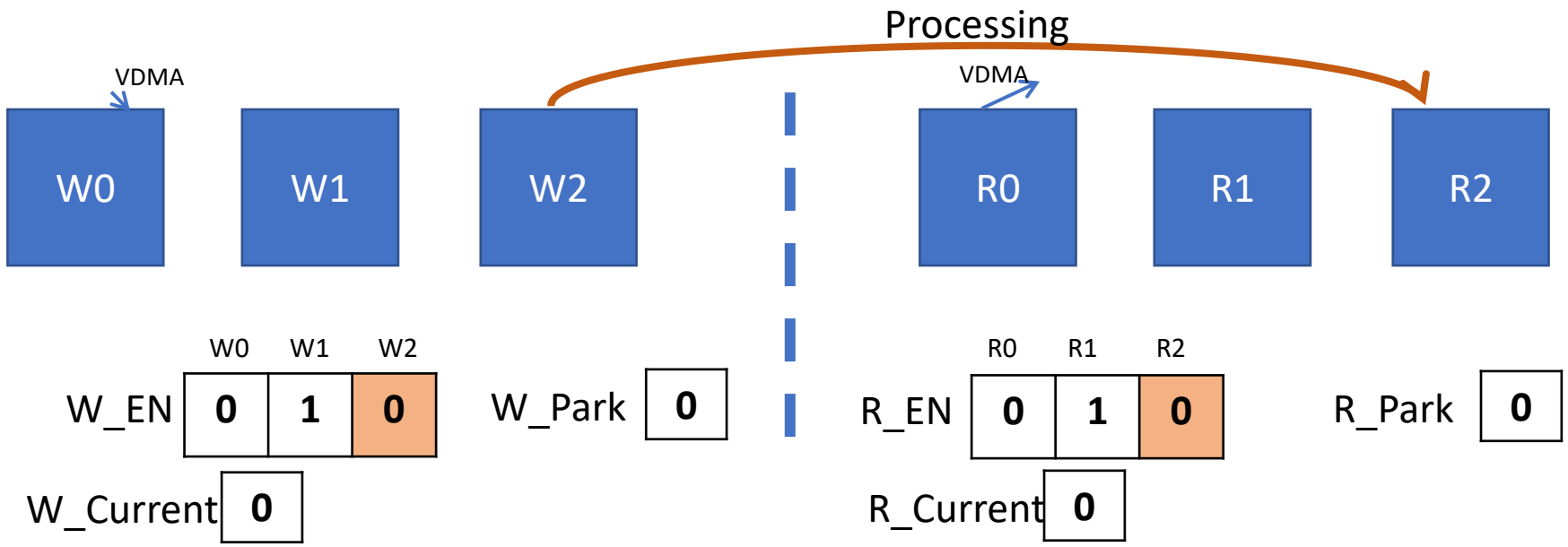
Processing się skończył ale W2 i R2 nie są gotowe więc czekamy



Kanał write zakończył odczytywanie ramki W2 i automatycznie przechodzi do zapisu ramki W0 (bo W_Park == 0) i wywołuje przerwanie. W callbacku sprawdzane jest czy W_EN[1] jest 0. Klatka jest wolna dlatego tylko inkrementujemy (zawsze modulo 3) W_Current



Kanał read zakończył odczytywanie ramki R2 i automatycznie przechodzi do odczytu ramki R0 (bo R_Park == 0) i wywołuje przerwanie. W callbacku sprawdzane jest czy R_EN[1] jest 0. R_EN[1] jest wolne dlatego nie parkujemy. R_Current inkrementujemy



Ponieważ zarówno W2 jak i R2 są wolne Processing przystępuje do działania

Errata

- Zakończenie processingu nastąpi tylko gdy następna ramka read i write jest gotowa. Spowodowane to jest tym że jeśli następna ramka read lub write nie jest dostępna to nie można zacząć processingu. Algorytm mógłby „przeskoczyć nad ramką którą chcemy procesować powodując złe działanie algorytmu.
- Processing kończy się gdy wszystkie dane z ramki zostaną przetworzone i zapisane do pamięci. Algorytm można przyspieszyć tym że odczyt nowej ramki może się zacząć przed zakończeniem zapisywania przetworzonych pikseli poprzedniej ramki do pamięci.

Splotowa filtracja 2d obrazu

Splotowa filtracja obrazu

Początek pamięci ramki np. W0

0	0	0	0	0	0	0	0
0	P0	P1	P2	P3	P4	P5	0
0	P6	P7	P8	P9	P10	P11	0
0	P12	P13	P14	P15	P16	P17	0
0	P18	P19	P20	P21	P22	S23	0
0	0	0	0	0	0	0	0

Wejściowy obraz z kamery (nieprzetworzony) o wymiarach 6x4

*

K0	K1	K2
K3	K4	K5
K6	K7	K8

Jądro filtru (Kernel) o wymiarach 3x3

=

Początek pamięci np. R0

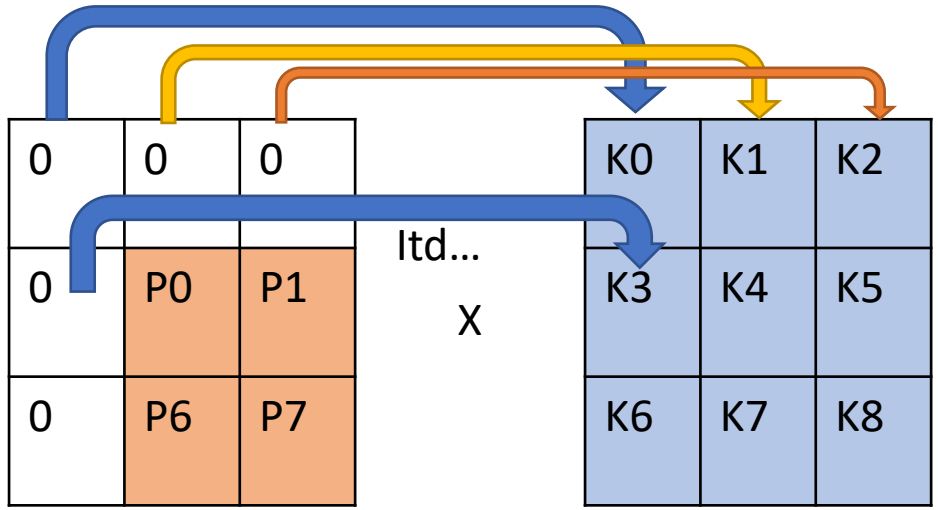
S0	S1	S2	S3	S4	S5
S6	S7	S8	S9	S10	S11
S12	S13	S14	S15	S16	S17
S18	S19	S20	S21	S22	S23

Przefiltrowany obraz o wymiarach 6x4 który zostanie przesłany na wyjście HDMI

- Zera dołożone dookoła ramki są po to aby możliwa była operacja splotu (zero-padding). W przypadku kernelu np. 5x5 potrzeba dodać 2 takie okręgi zer i tak dalej. Kernel musi mieć być kwadratowy i mieć wielkość nieparzystą.
- Przed uruchomieniem algorytmu następuje wyzerowanie obszarów pamięci na których zarezerwowane są miejsca dla W0,W1 i W2.
- Dzięki funkcji stride w VDMA można ustawić mniejszy obszar dla pisania niż wielkość pamięci W0 przez co możliwe jest zrobienie takiej obwoluty zer.

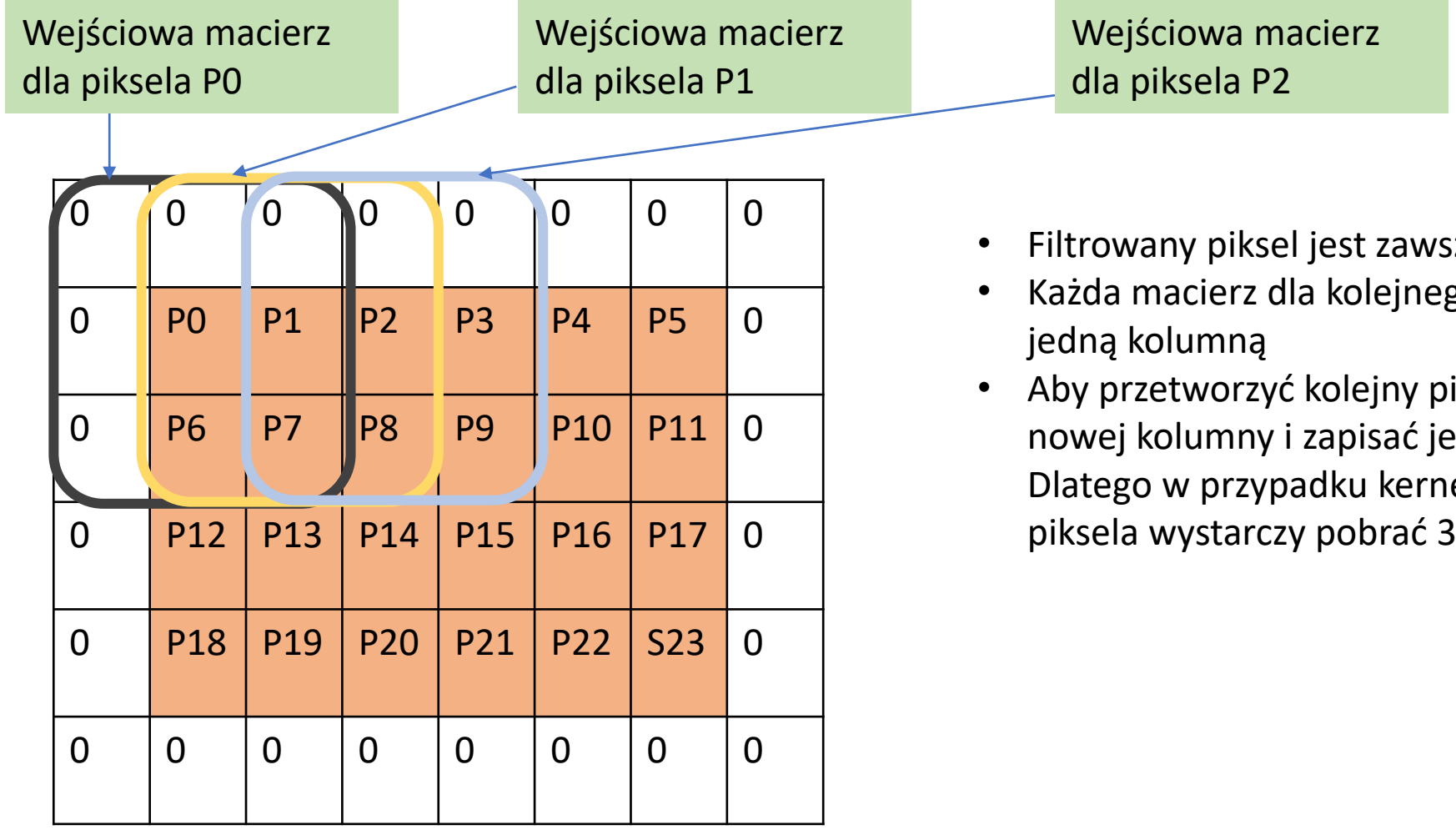
Przykład filtracji piksela P0

0	0	0	0	0	0	0	0
0	P0	P1	P2	P3	P4	P5	0
0	P6	P7	P8	P9	P10	P11	0
0	P12	P13	P14	P15	P16	P17	0
0	P18	P19	P20	P21	P22	S23	0
0	0	0	0	0	0	0	0



S0 = K0*0 + K1*0 + K2*0 + K3*0 + K4*P0 + K5*P1 + K6*0 + K7*P6 + K8*P7

Wartość przefiltrowanego piksela P0



- Filtrowany piksel jest zawsze na środku wejściowej macierzy
- Każda macierz dla kolejnego piksela w wierszu różni się tylko jedną kolumną
- Aby przetworzyć kolejny piksel wystarczy pobrać wartości dla nowej kolumny i zapisać je w miejscu już niepotrzebnych pikseli. Dlatego w przypadku kernela 3x3 dla przetworzenia kolejnego piksela wystarczy pobrać 3 piksele

Jak to przyspieszyć ?

Wejściowa macierz dla piksela P0

0	0	0	x
0	P0	P1	x
0	P6	P7	x

0
P2
P8

Następna filtracja

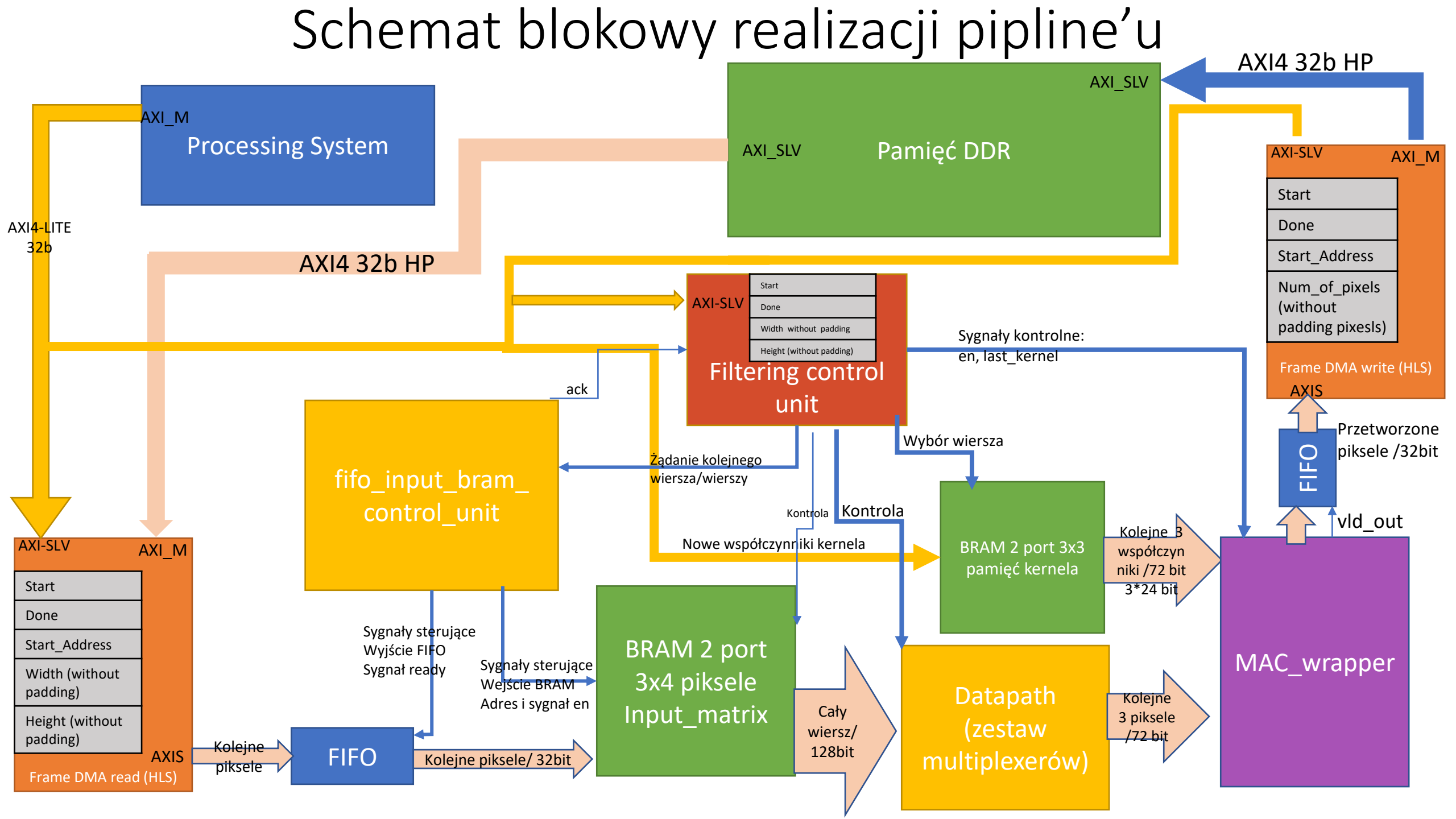
Wejściowa macierz dla piksela P1

x	0	0	0
x	P0	P1	P2
x	P6	P7	P8

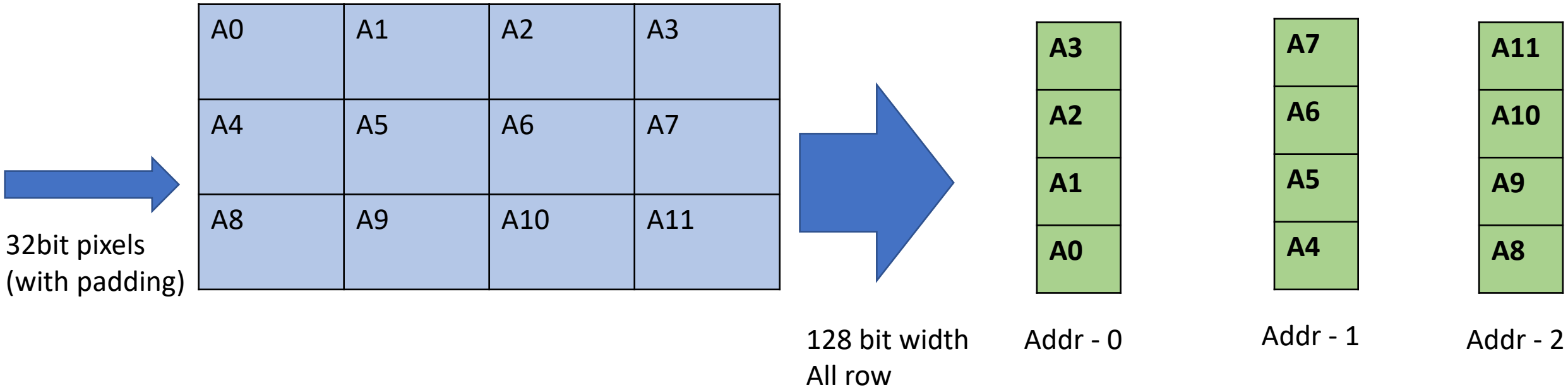
W momencie filtracji piksela P0 przesyłana jest brakująca kolumna dla filtracji piksela P1

- Na początku każdej nowej linii wymagane jest zapisanie 3 kolumn potem wystarczy tylko 1 i tak aż do końca linii.

Schemat blokowy realizacji pipline'u

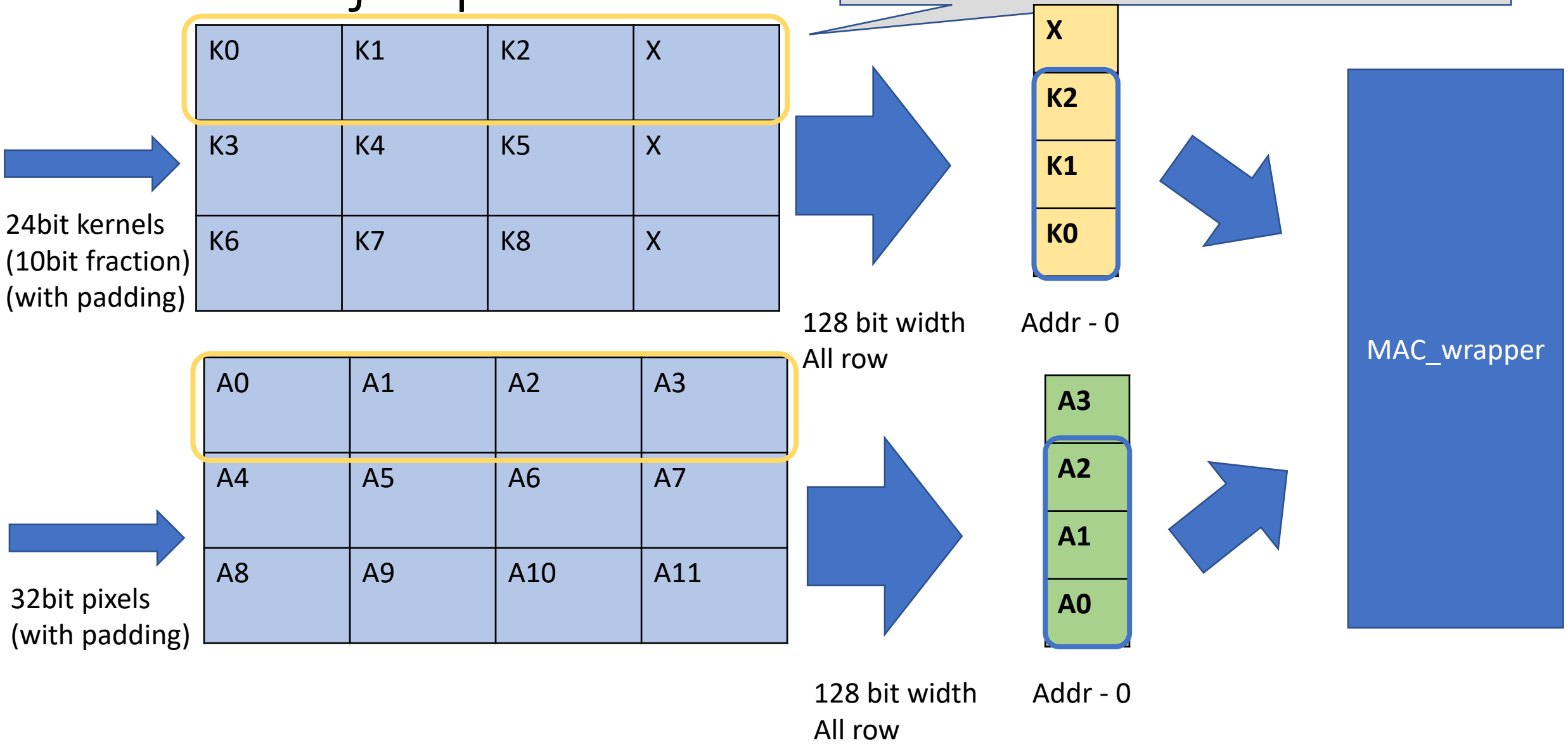


Bram 2 port input_matrix

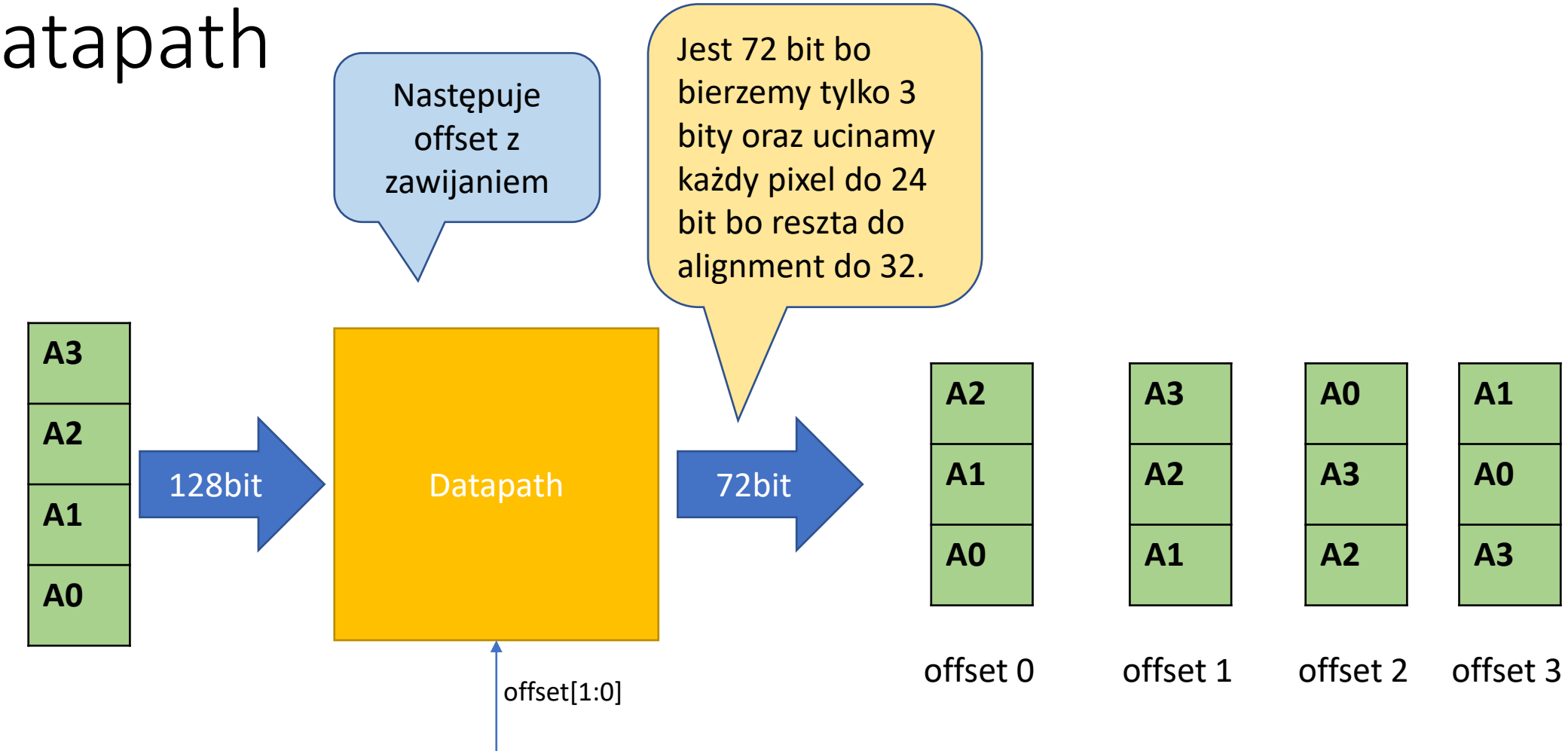


Realizacja Splotu - bram

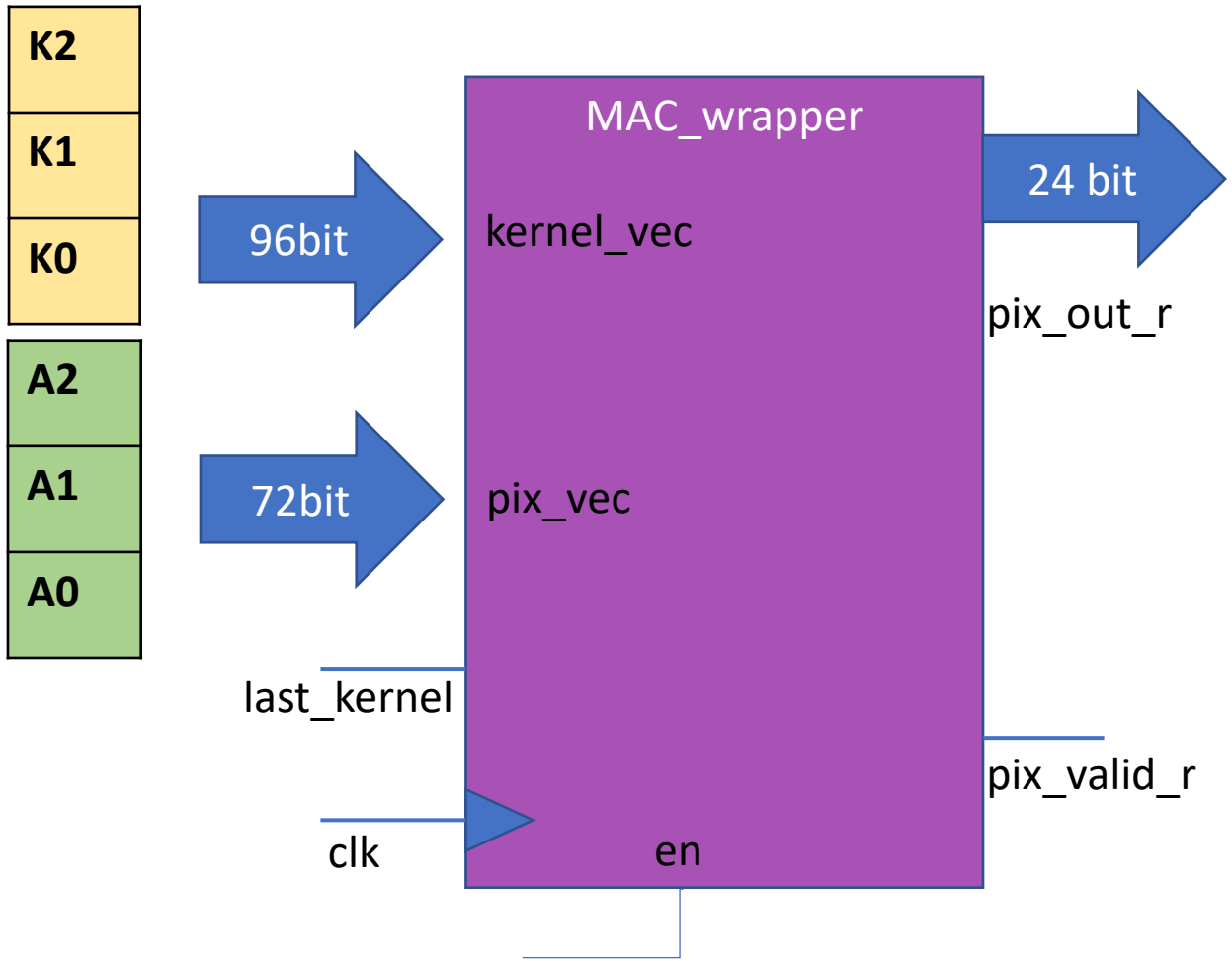
Musi pamięć do krenela być 4x4 ponieważ gdyby było tylko 3x3 to nie dało by się na wyjściu zrobić 3*32bit tylko 1 *32 bit więc jest 4*32 bit



Datapath


Następuje offset z zawijaniem
Jest 72 bit bo bierzemy tylko 3 bity oraz ucinamy każdy pixel do 24 bit bo reszta do alignment do 32.
A3
A2
A1
A0
128bit
Datapath
72bit
offset[1:0]
A2
A1
A0
A3
A2
A1
A0
A3
A2
A1
A3
A0
offset 0
offset 1
offset 2
offset 3

MAC_wrapper

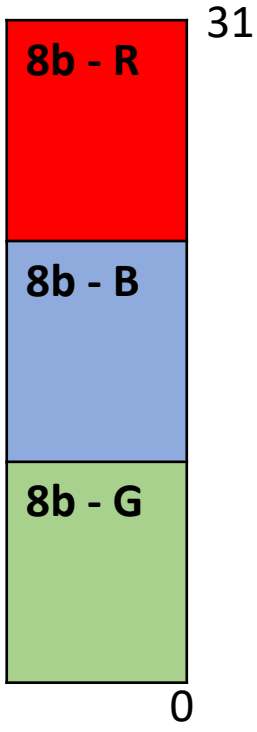


W tym module następuje przemnożenie pikseli przez odpowienią wartość kernela i akumulacja w rejestrze 32 bitowym. Kernele będą 24 bitowe (10 bitów ułamkowa część), wchodzą 32 bity bo bram musi być 32 bit żeby pod axi podpiąć na wejściu są ucinane bity wyższe. Podczas narastającego zbocza clk gdy last_kernel jest 1'b1, następuje przemnożenie ostatnich kerneli przez piksele i akumulacja. Ponadto dane są przenoszone do następnego stopnia pipline'u gdzie następuje sumowanie częściowych produktów mnożeń. Gdy dane na wyjściu są gotowe na 1 takt pojawia się sygnał pix_valid_r który steruje fifo. (założenie fifo nie może być nigdy pełne !!). Na końcu pipline'u zostaje z 32 bitów signed fixed-point obcięcie do 24 bit unsigned (można zaimplementować np. zaokrąglenie itp.). Dane wejściowe są przyjmowane tylko gdy na wejściu en jest 1.

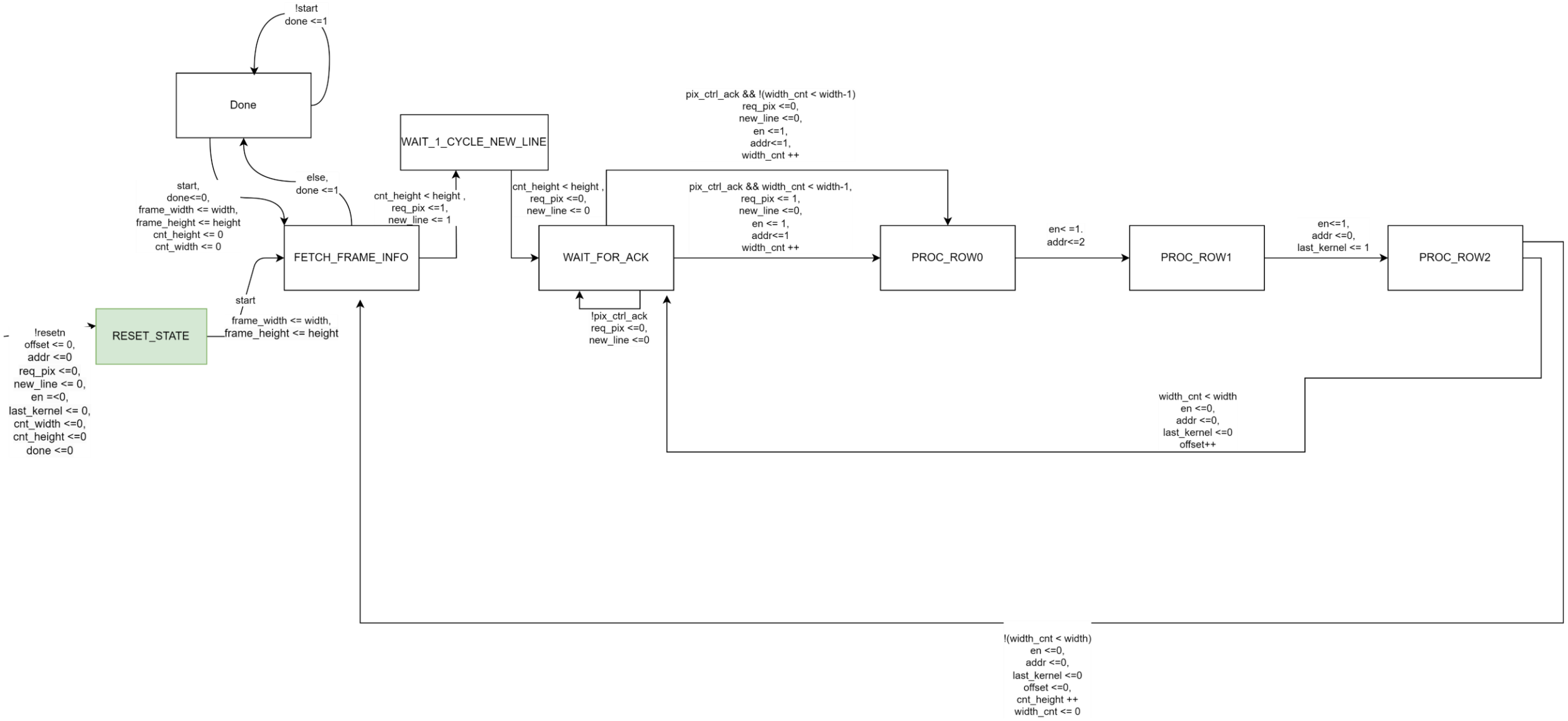
Errata - datapath

- Datapath wykazywał duże czasy propagacji przez co nie można było podkręcić pipline’u do 150 MHz. Dlatego zdecydowano się przed każdym wejściem do MAC_wrapper dodać przerzutnik przez co linia kombinacyjna została skrócona i możliwe było uzyskanie 150 MHz.

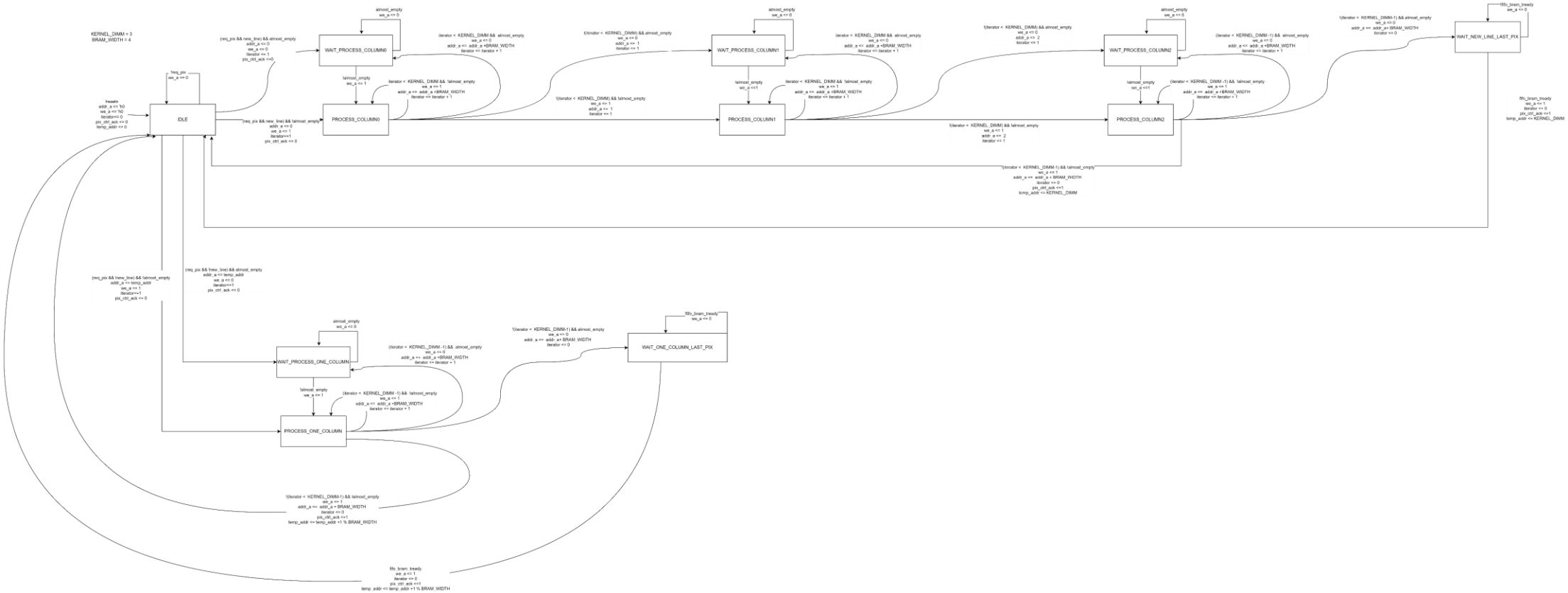
Rozkład bitów w wektorze pikseli



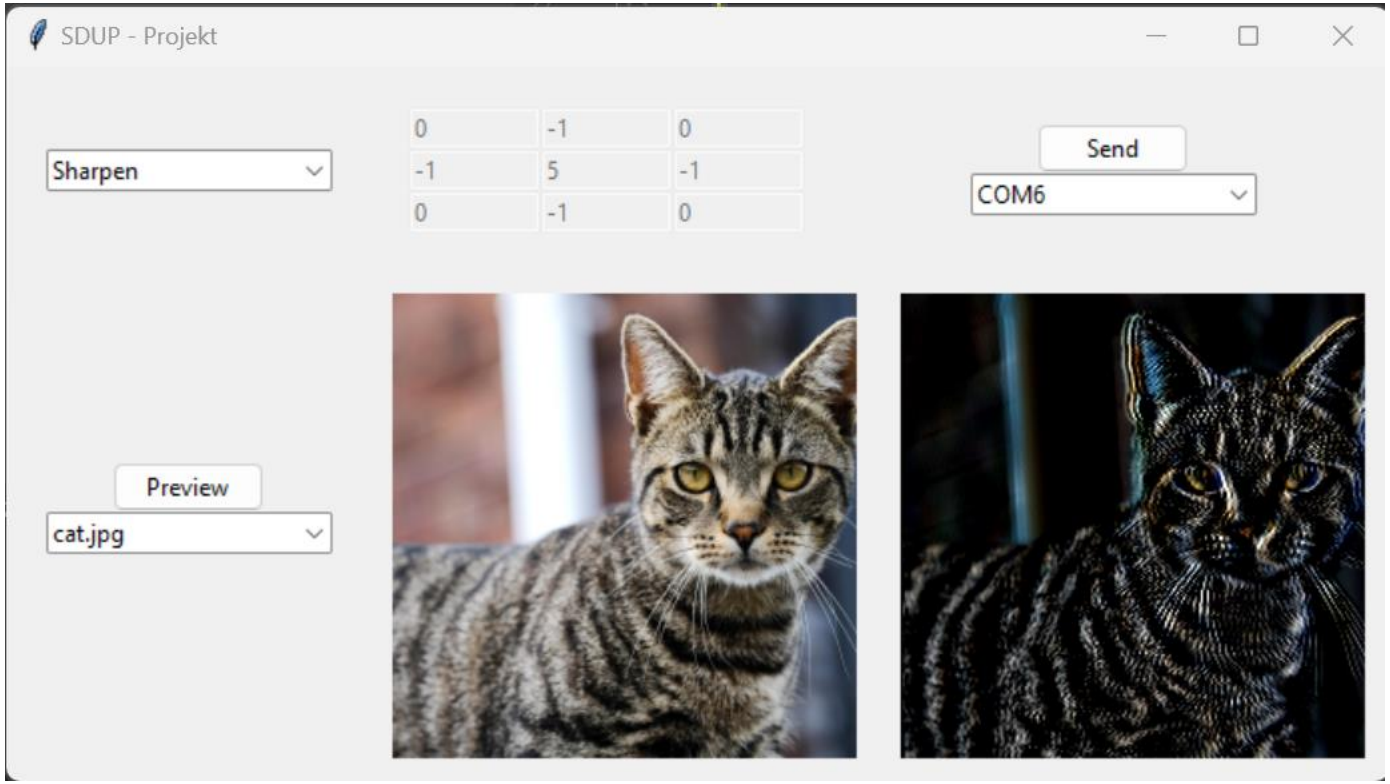
FSM filtering control unit



FSM- fifo_input_bram_control_unit



Aplikacja GUI do obsługi filtracji



Wykorzystanie zasobów

Resource	Utilization	Available	Utilization %
LUT	14221	53200	26.73
LUTRAM	698	17400	4.01
FF	16761	106400	15.75
BRAM	17	140	12.14
DSP	15	220	6.82
IO	24	125	19.20
BUFG	7	32	21.88
MMCM	2	4	50.00