POLITECHNIKA POZNAŃSKA

Wydział Automatyki, Robotyki
i Elektrotechniki

Jakub Wicher
147589

Adrian Szymankiewicz
147568

Mateusz Koza
147606

DOKUMENTACJA PROJEKTU „STATKI”

Zespół projektowy G5-5

Grupa L11

8 sierpnia 2022

1. OMÓWIENIE ZREALIZOWANEGO TEMATU

Tematem projektu jest gra oparta na kultowej mechanice statków. Celem graczy jest w jak najkrótszym czasie zniszczyć statki przeciwnika. W początkowej fazie rozgrywki umieszczają oni swoje statki na matrycy 12 na 12, która następnie jest rozstrzeliwania przez drugą osobę. Mają do dyspozycji statki o wielkościach: (4x1)x1, (3x1)x2, (2x1)x3, (1x1)x4. Celują oni „na ślepo". Jedyną informacją zwrotną na matrycy trafień jest oznaczenie nietrafiony, trafiony – na sąsiednich polach znajdują się kolejne części tego statku oraz zatopiony. Aktualny ruch gracza określa strzałka, w tym czasie druga matryca jest zablokowana. Ostateczny wynik jest określany przy pomocy czasu rozgrywki (chwila zatopienia ostatniego statku na wygranej matrycy).

2. PODZIAŁ PRAC W ZESPOLE PROJEKTOWYM

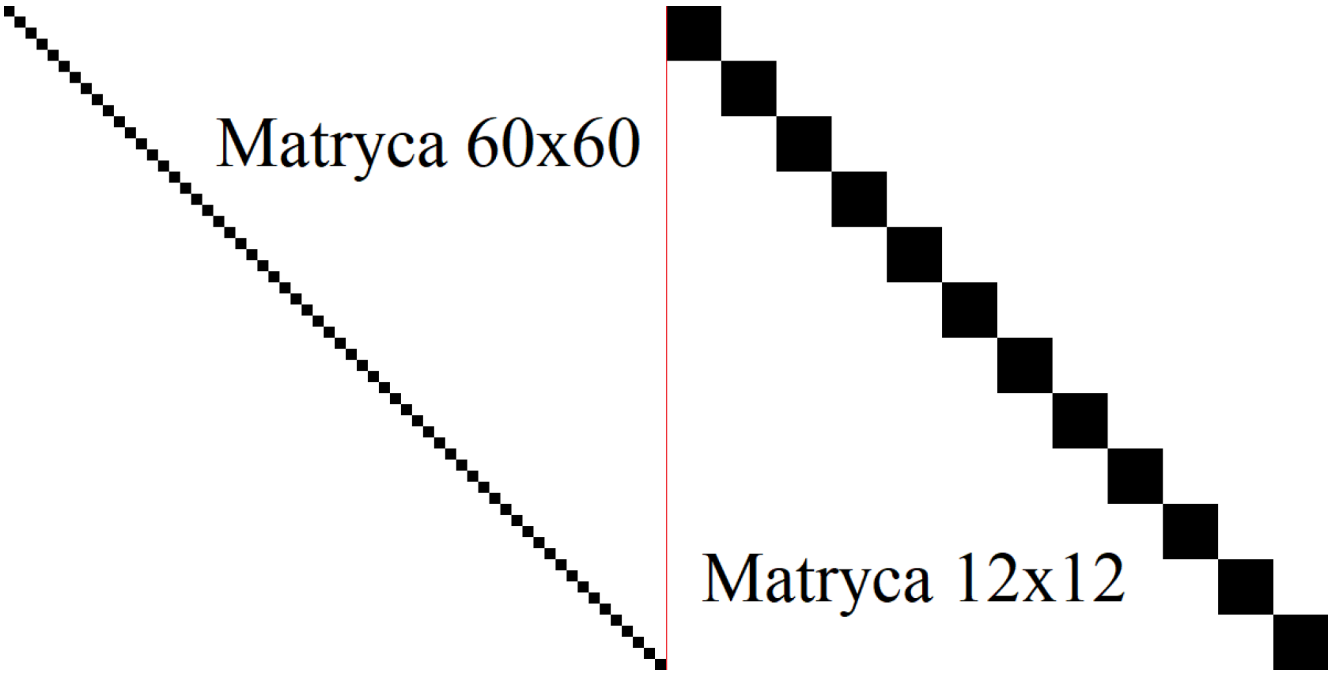
- Jakub Wicher
  - app.circ
  - cursor_module.circ
  - main.circ
  - memory_management.circ
  - screen_xy_to_screen_data.circ
  - secs_to_score.circ
  - ship_data_to_all_ship_xy.circ
  - ship_out_of_map.circ
  - ship_xyr_to_screen_xy.circ
  - ships_colliding.circ
  - timer.circ
- Adrian Szymankiewicz
  - dokumentacja
  - 4bit_greater_or_equal_than_and_less_than.circ
  - 7seg_disp_driver.circ
  - arrow.circ
  - demuxor_30x30x3.circ
  - drawing_select.circ
  - keyboard_input.circ
- Mateusz Koza
  - bot_0.circ
  - bots.circ
  - map_generator.circ
  - not_hit_module.circ
  - not_hit_ram_storage.circ
  - random_num_generator_A.circ
  - random_num_generator_B.circ
  - shooter.circ
  - tick.circ
  - tick2.circ

3. FUNKCJONALNOŚCI UKŁADU STATKI

- Matryca 60x60 i druga gdzie oddano strzał
- Przeciwnik komputerowy o jednym stopniu trudności
- Ranking, czas rozgrywki
- Obsługa klawiatury
- Granie z kimś na żywo (siedzącym obok)
- Losowanie statków

4. OMÓWIENIE KAŻDEGO ZE ZREALIZOWANYCH MODUŁÓW

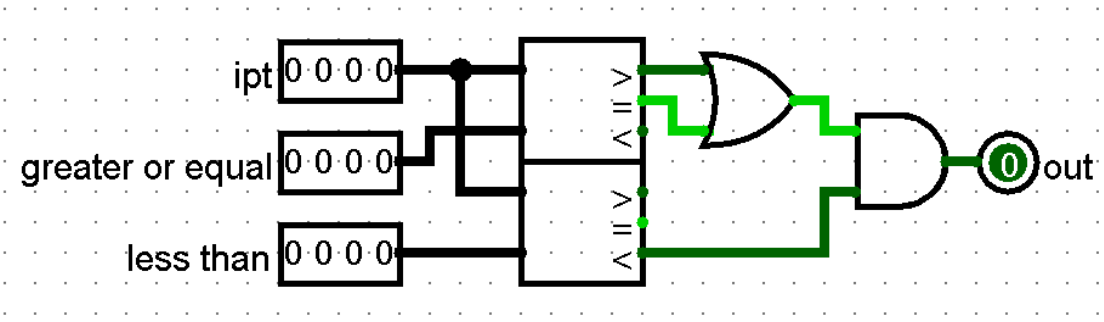
Założenie podstawowe: wprowadzono dwa sposoby obsługi ekranu. Jeden o indeksach od 0 do 59 dzielący matryce na 60 części i od 0 do 11 dzielący matrycę na 12 części – kwadraty 5 na 5 pikseli, są to pola na których porusza się kursor.



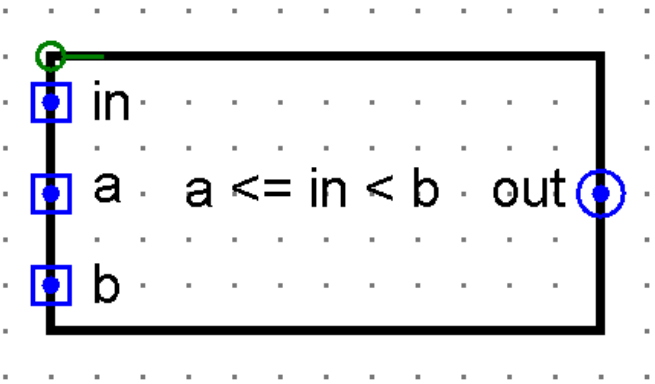
Rys. 1. Graficzne przedstawienie używanych typów podziału ekranu

4.1. 4bit_greater_or_equal_than_and_less_than.circ

Układ ograniczający 4-bitowe wejście do narzuconych wartości ograniczeń logicznych (jednocześnie większe równe od x oraz mniejsze od y). Jeśli wejście znajduje się w zakładanym zakresie układ zwraca logiczną prawdę, jeśli nie logiczny fałsz. Posiada trzy wejścia (liczba porównywana, i dwa ograniczniki) oraz jedno wyjście (czy warunek logiczny jest spełniony). Składa się z dwóch komparatorów, bramki OR oraz bramki AND.



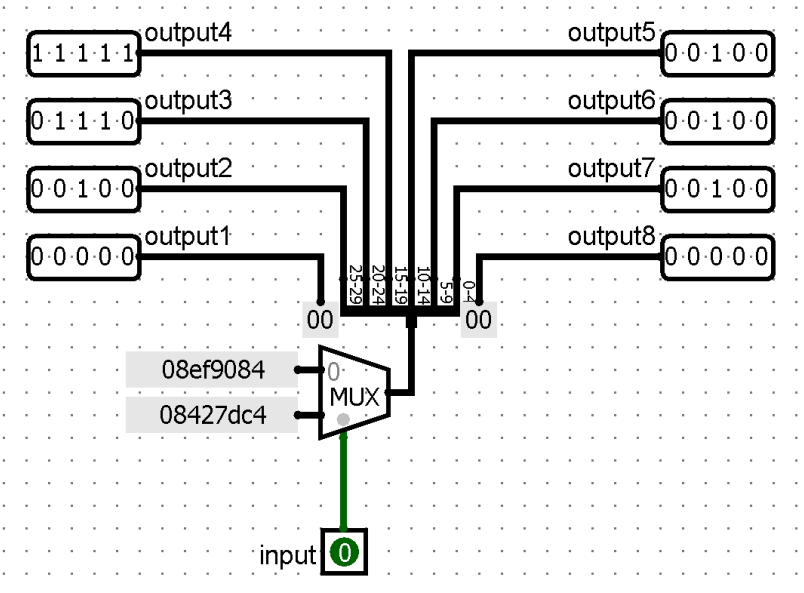
Rys. 2. Układ porównująco-ograniczający wejście (look under mask)



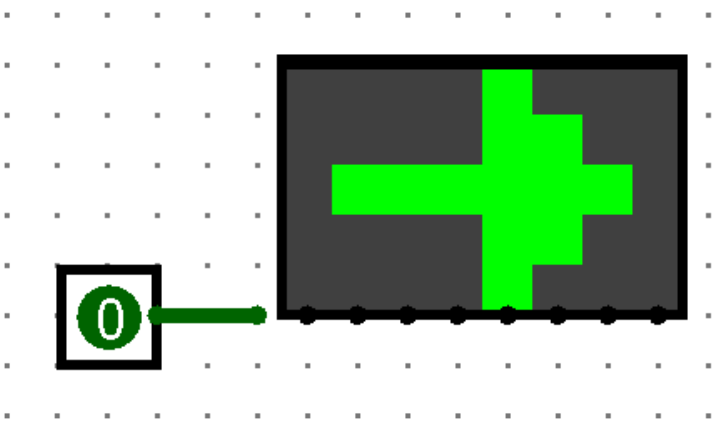
Rys. 3. Blok układu porównująco-ograniczający wejście

4.2. arrow.circ

Prosty układ pozwalający, poprzez zmianę stanu jednobitowego wejścia, otrzymać na wyjściu dwa różne obrazy. Wykorzystany został do określenia tego, który gracz może aktualnie wykonać ruch. Zbudowany jest z zakodowanej strzałki w kodzie hexagonalnym o dwóch orientacjach następnie poprzez zmianę stanu wejścia sterującego multipleksera dokonywany jest wybór wyświetlania aktualnego obrazu.



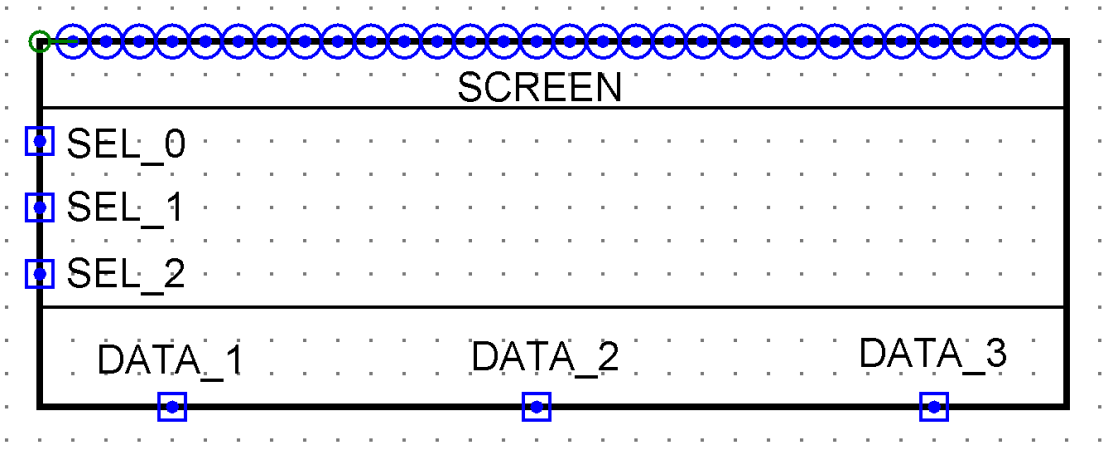
Rys. 4. Układ do określenia kolejki gracza (look under mask)



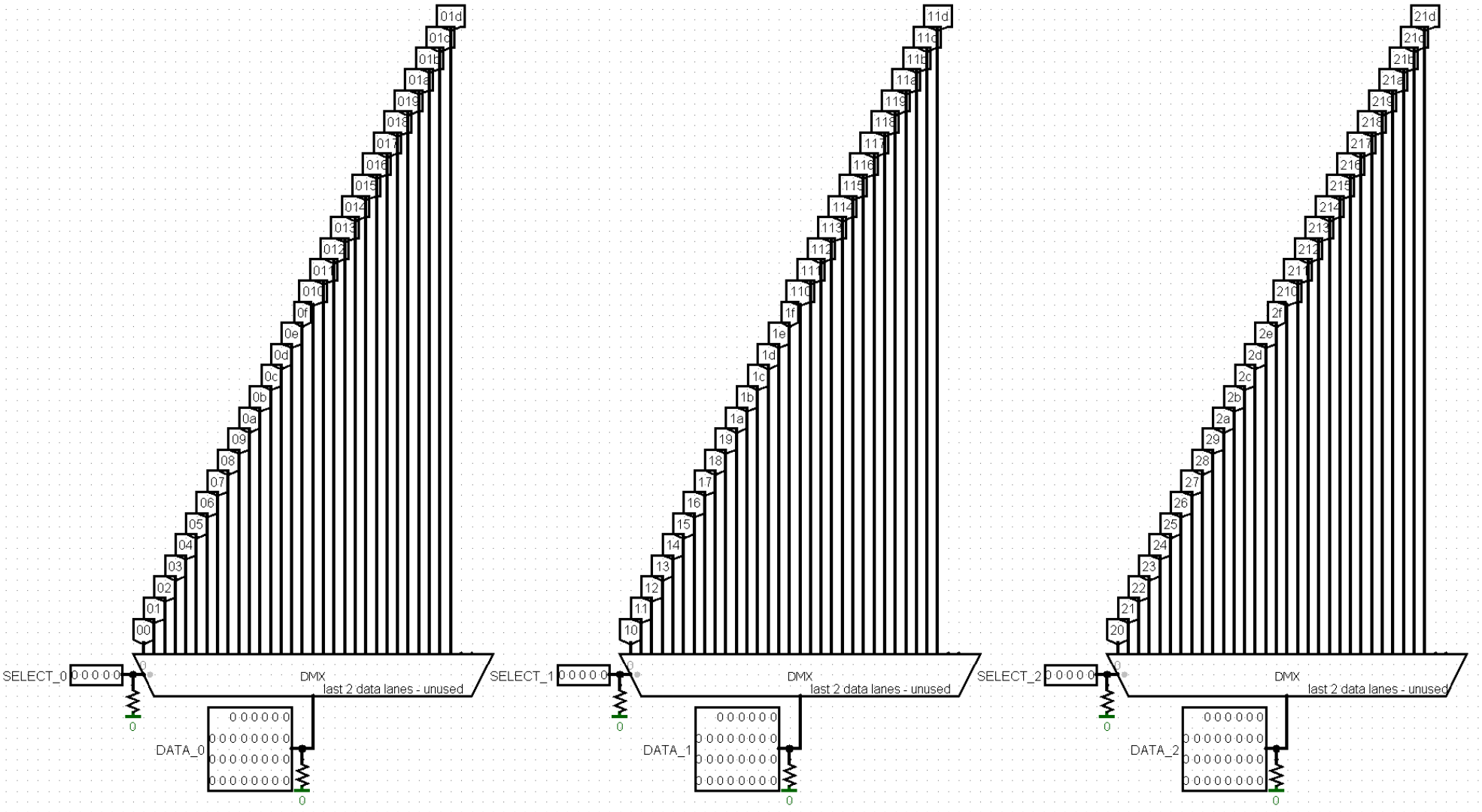
Rys. 5. Graficznie przygotowany układ pokazuje (wyświetla strzałkę w odpowiednią stronę) kolejkę gracza

4.3. demuxor_30x30x3.circ

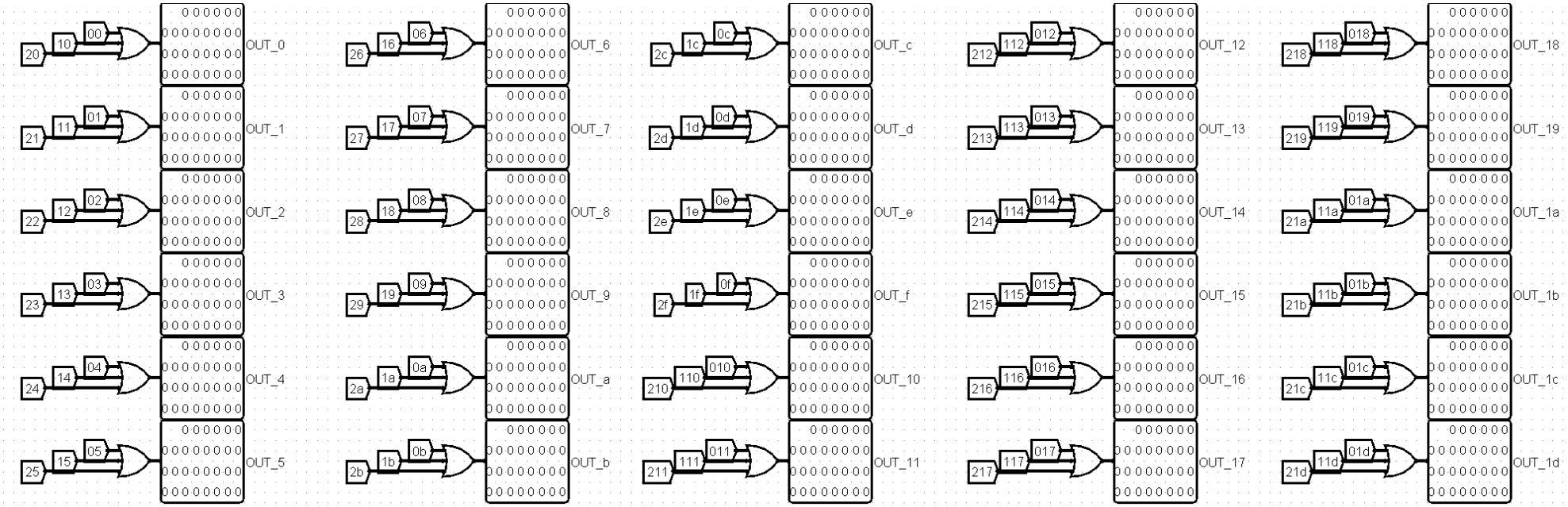
Układ łączący wyjścia układu screen_xy_to_screen_data.circ. Pozwala nadpisywać na sobie dane graficzne wygenerowane przy pomocy tego bloku. Każdy sygnał wychodzący z demultipleksera na wejściu (przedstawionego na Rys. 7.) wchodzi do bramki logicznej OR i dalej idzie finalne na jedną kolumnę ekranu. Blok na Rys. 6. na wejściu otrzymuje sygnały SELECT obsługujące demultipleksery oraz sygnały danych uzyskane z układu screen_xy_to_screen_data.circ. Na wyjściu natomiast podłączany jest ekran.



Rys. 6. Graficznie przygotowany układ nadpisujący na siebie wygenerowane dane graficzne



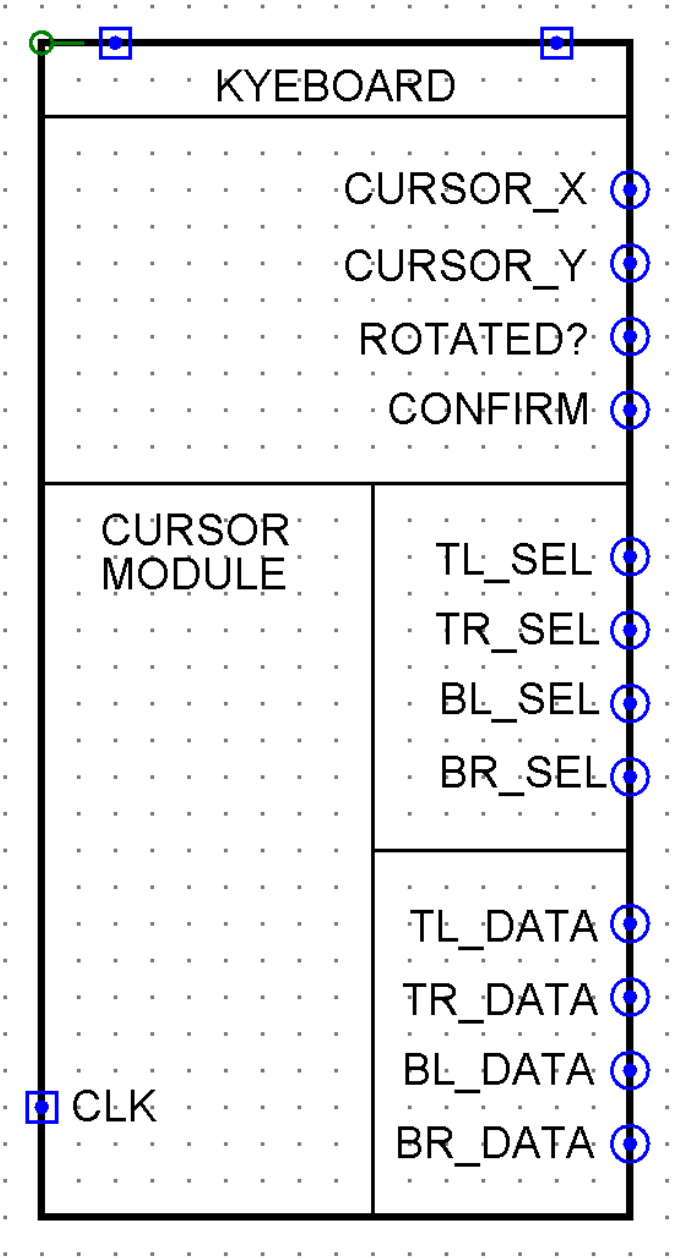
Rys. 7. Układ nadpisujący na siebie wygenerowane dane graficzne (look under mask) część 1.



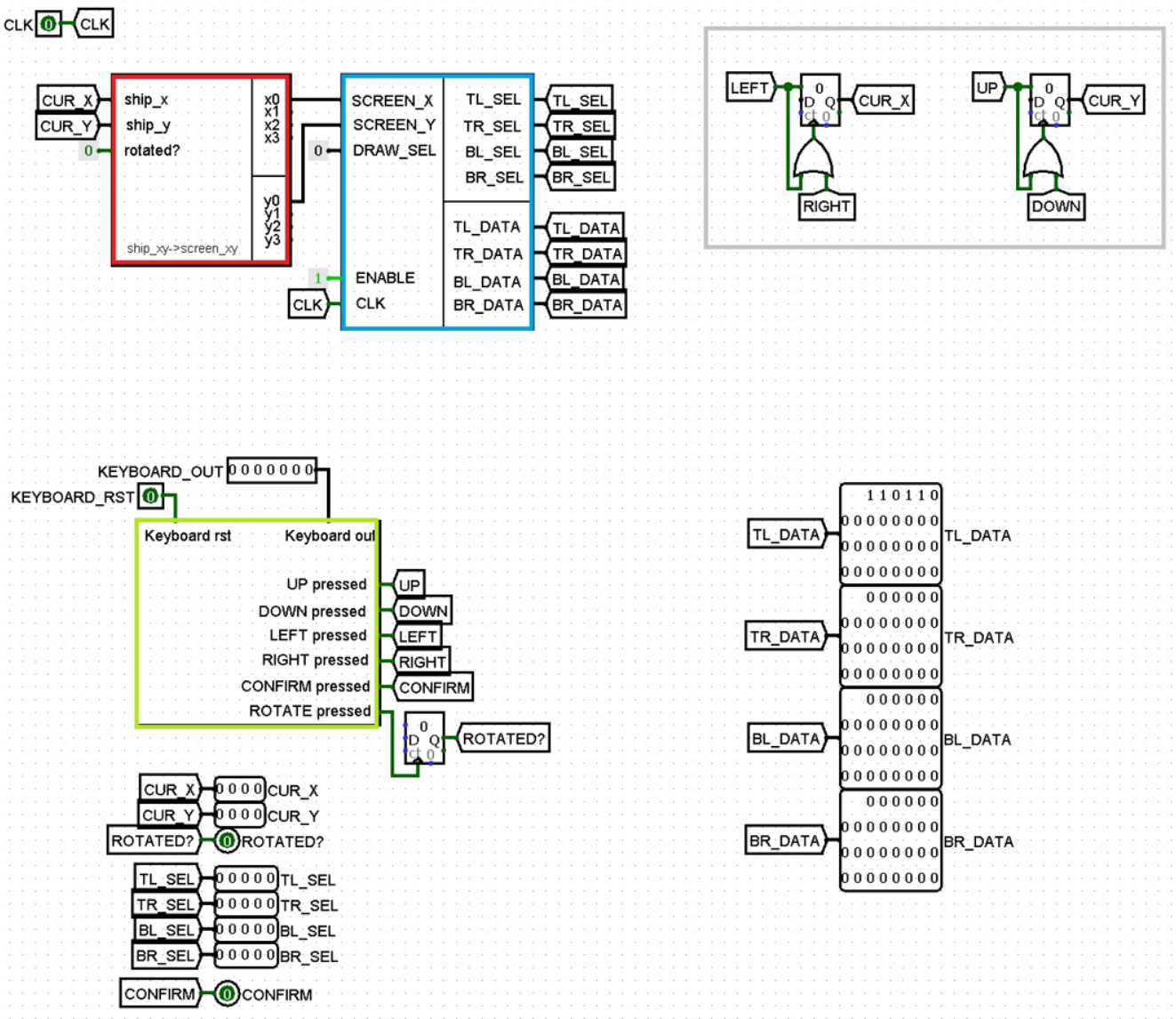
Rys. 8. Układ nadpisujący na siebie wygenerowane dane graficzne (look under mask) część 2.

4.4. cursor_module.circ

Połączenie trzech mniejszych układów, wspólnie realizujących funkcję pełnej obsługi ruchu i aktywacji kursora z poziomu klawiatury. Blok tego układu przedstawiony na Rys. 9. posiada dwa wejścia obsługujące klawiaturę (input, output, klawiatura przekazuje 7-bitową liczbę awsd i enter w tym przypadku) Wyjścia CURSOR_X, CURSOR_Y oraz CONFIRM określają aktualne położenie i aktywację (0 lub 1) kursora. Wyjście stan wyjścia ROTATED? określa czy statek jest obrócony czy też nie (statek nieobrócony jest poziomy). Natomiast wyjścia poniżej tego służą jedynie do wyświetlania kursora na ekranie.



Rys. 9. Zamknięty blok układu cursor_module.circ

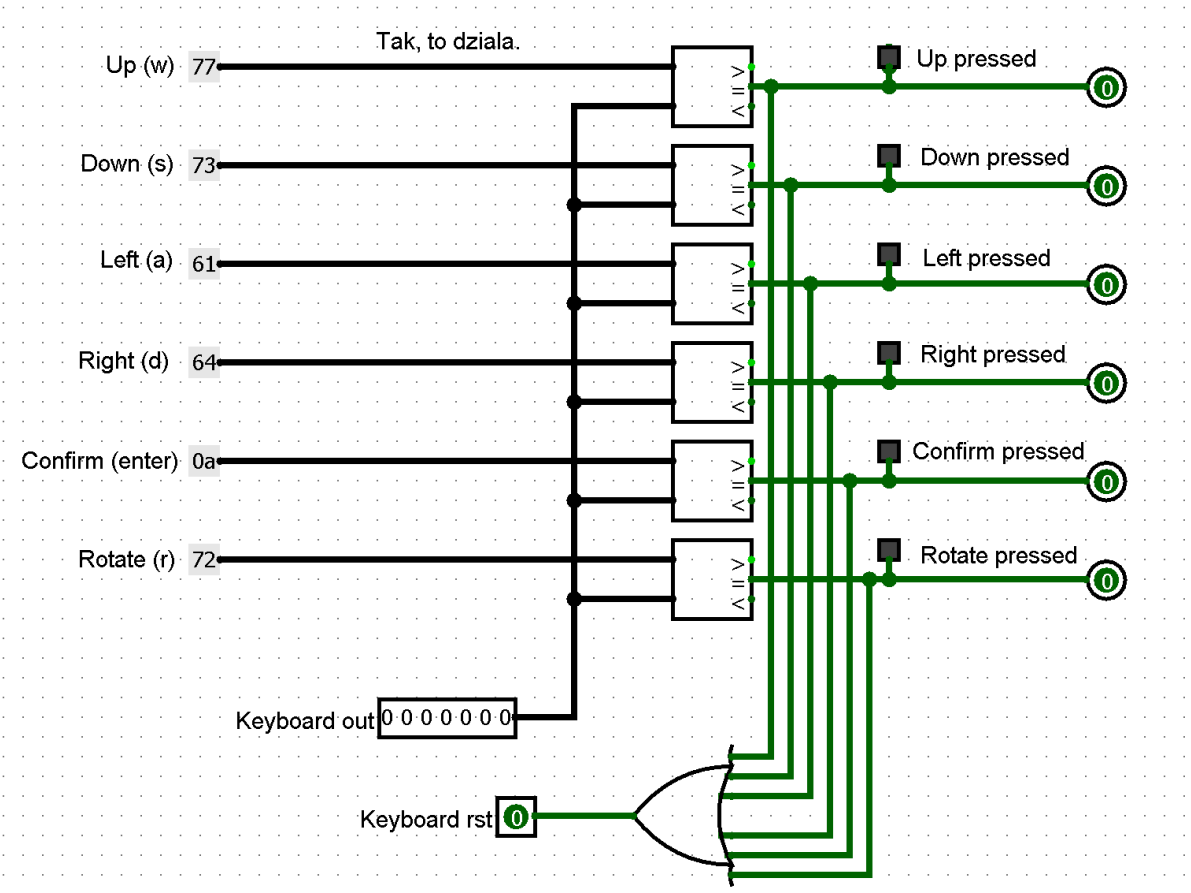


Rys. 10. Układ cursor_module.circ (look under mask), bloki nieopisane służą jedynie do obsługi labeli

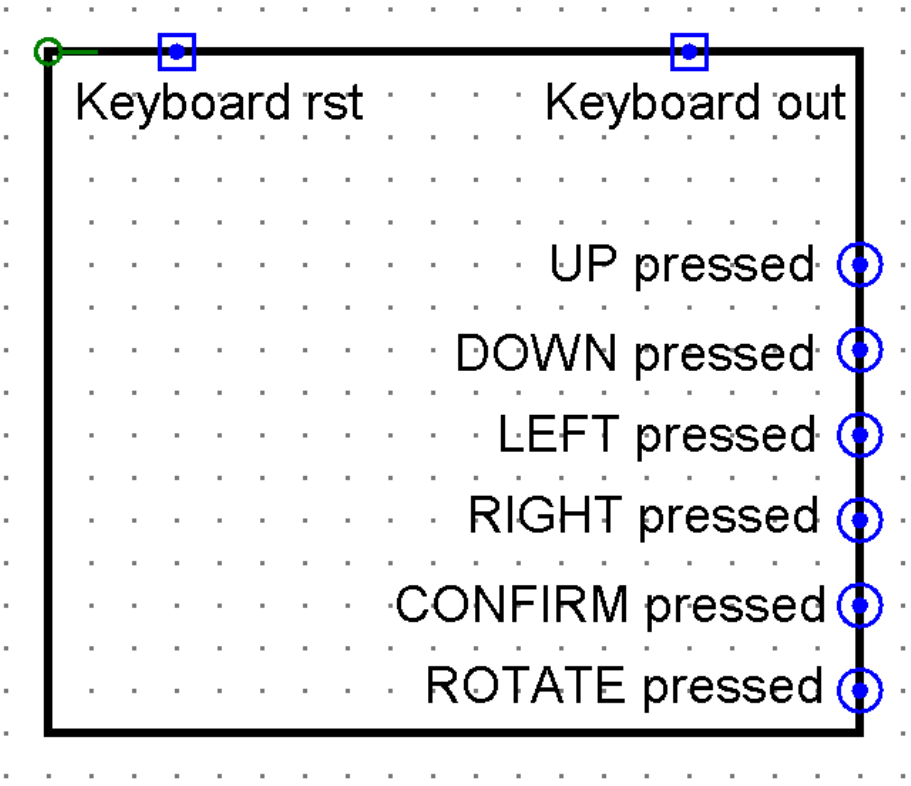
Część układu oznaczona kolorem szarym zapewnia zapis aktualnej pozycji kursora, oś y układu współrzędnych ekrany jest zwrócona w dół przez co aby przesunąć kursor w dół, licznik musi odliczać również w dół. Klikając w górę licznik staje się dekrementalny (ponieważ dostaje 1 na wejście), zacznie od końca, więc ruch zostanie wykonany poprawnie.

4.4.1 keyboard_input.circ

Układ zapewniający obsługę klawiatury, oznaczony kolorem zielonym na Rys. 10.. Podstawą działania tego układu są komparatory, przyrównujące stałą wartość liczbową, która przedstawia odpowiednie wejście z klawiatury za pomocą kodu ASCII (np. w - 87 (dec) - ... (hex)). Reset realizowany jest przy pomocy bramki OR, znaki na klawiaturze automatycznie zapisują się w kolejkę, dzięki temu przy wciśnięciu jednego znaku, dopóki nie zostanie on przetworzony (cały sygnał zegara się nie zmieni), nie można klikać innych. Wejścia - obsługa sześciu przycisków klawiatury, wyjścia stan logiczny 1/0 oznaczający wciśnięcie lub jego brak.



Rys. 11. Układ do obsługi klawiatury (look under mask)

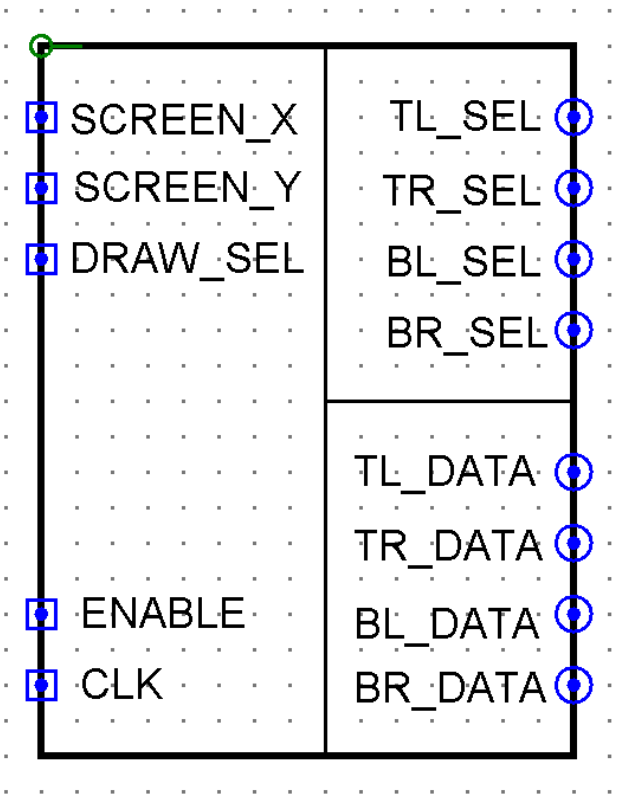


Rys. 12. Układ do obsługi klawiatury

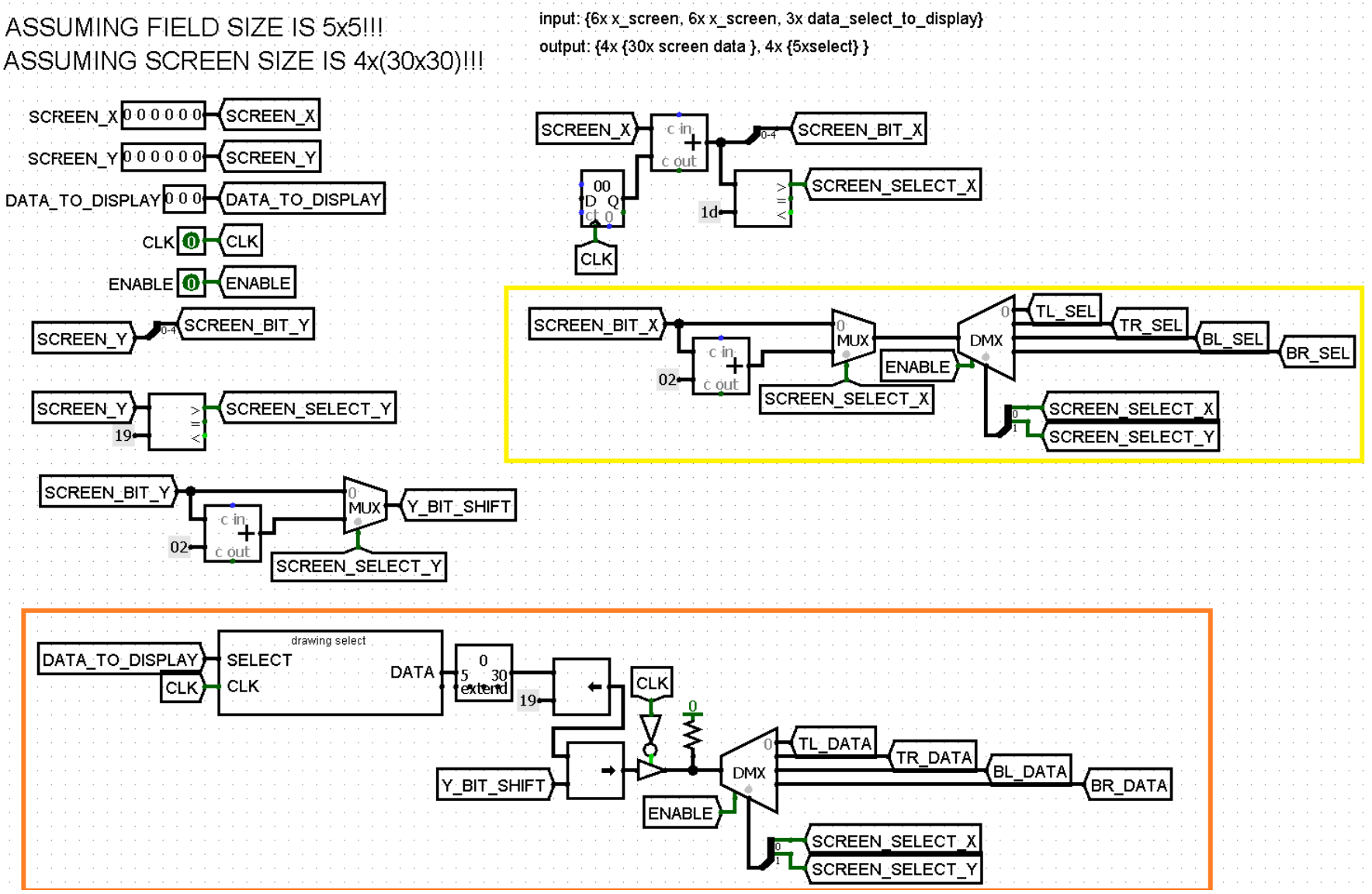
Niebieski moduł z Rys. 10. potrzebuje koordynatów z pełnego ekrany dlatego na jego wejście podłączony został blok ship_xyr_to_screen_xy.circ wykonujący to zadanie. Używane są natomiast tylko jego pierwsze współrzędne, bo reszty kursor i tak nigdy nie będzie miał. Ponadto jego wejście na jego wejście ENABLE podawany jest stale sygnał wysoki utrzymujący jeden obrazek z modułu drawing_select.circ.

4.4.2. screen_xy_to_screen_data.circ

Układ ma za zadanie przekonwertować lokalizacje na ekranie do danych na podpisywanych na ekran. Jest to konieczne, ponieważ rozłożenie danych na dwa układy współrzędnych (jeden związany z całym ekranem, a drugi dzieląc ekran na pola dwunastu koordynatów. Umożliwia to łatwiejszą obsługę samej gry. Schematyczny blok układu został przedstawiony na Rys. 13.. Wejścia SCREEN_X oraz SCREEN_Y określają położenie na docelowym ekranie, a DRAW_SEL wybór grafiki z bloku drawing_select.circ. Wejście ENABLE albo przekazuje na ekran to co jest podawane przez wejście DRAW_SEL, albo daje same zera - jest potrzebne do zmiany rysunku. CLOCK umożliwia natomiast wyświetlanie po kolei kolumn grafiki. Natomiast wyjścia bloku definiują wybory oraz użyte pamięci przypisane do poszczególnych ekranów (jeden z czterech) tj. TL – top left, TR – top right, BL – bottom left, BR – bottom right.



Rys. 13. Układ screen_xy_to_screen_data schemat bloku



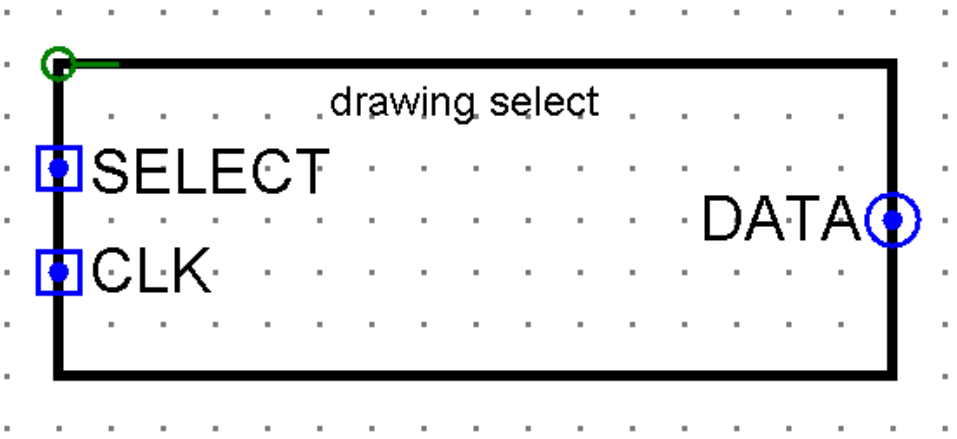
Rys. 14. Układ screen_xy_to_screen_data (look under mask)

Na Rys. 14. od lewego górnego rogu idąc w dół wprowadzono potrzebny input. Następnie ograniczono sygnał SCREEN_Y jedynie do pięciu pierwszych bitów, ponieważ w tym miejscu potrzeba jedynie zakresu do 32, i w tym momencie najbardziej znaczący bit jest nieistotny (natomiast dalej, w tym samym układzie, jest od konieczny do określenia na którym ekranie znajduje się kursor. Dalej porównano tę wartość do 25 w celu obsłużenia przechodzenia na kolejny ekran (wtedy to się dzieje). Następnie do 5 bitów dodano jeden (dla x_pos 1 i y_pos 1), który został pominięty wcześniej oraz oba sygnały, pierwotny i sumę, przepuszczono przez multiplekser mający na wejściu SELECT sygnał wyboru ekranu do wyświetlania (niweluje to problem braku przeskakiwania kursora na kolejny ekran).

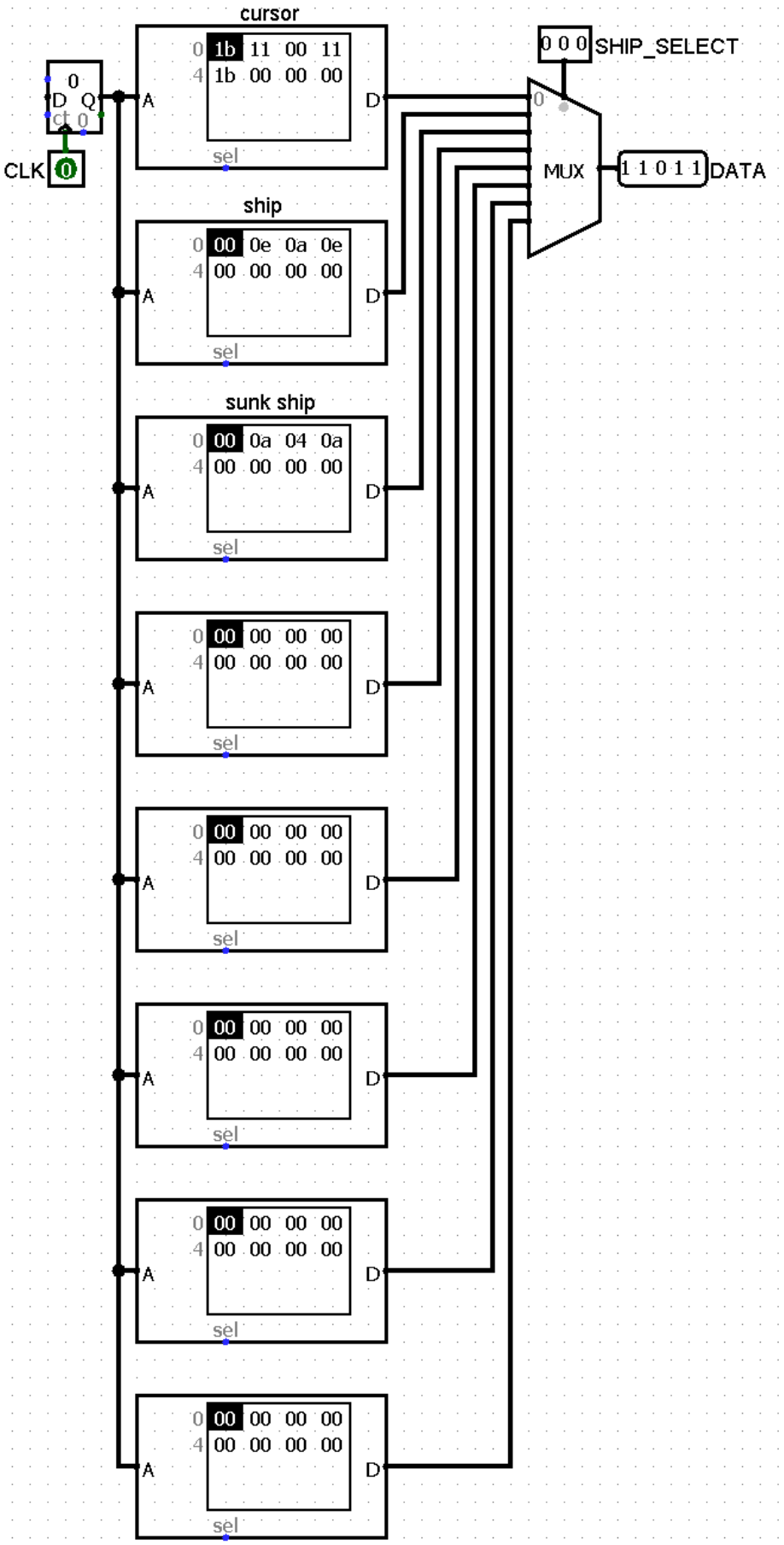
Kolorem pomarańczowym oznaczono układ, na którego wejściu pobierany jest sygnał z bloku drawing_select.circ.

4.4.2.1. drawing_select.circ

Układ zapewniający wyświetlenie w danym miejscu obrazka wcześniej zapisanego w pamięci ROM. Na Rys. 15. wejście SELECT pozwala wybrać potrzebną aktualnie pamięć (wybór pamięci ROM z drzewka na Rys. 16.). Sygnał CLK podawany do licznika, służy jedynie do przeskakiwania na kolejne komórki każdej z pamięci, aby wyświetlić odpowiednią kolumnę obrazka w danym momencie. następnie wszystkie sygnały przechodzą równolegle przez jeden multiplexer, w którym przy pomocy wejścia SELECT wybierany jest potrzebny typ pamięci. Wyjściem układu jest 5-bitowy sygnał DATA określający w danej chwili jedną kolumnę wyjściowego obrazu.



Rys. 15. Graficznie przygotowany układu pozwalającego na wybór danych do narysowania

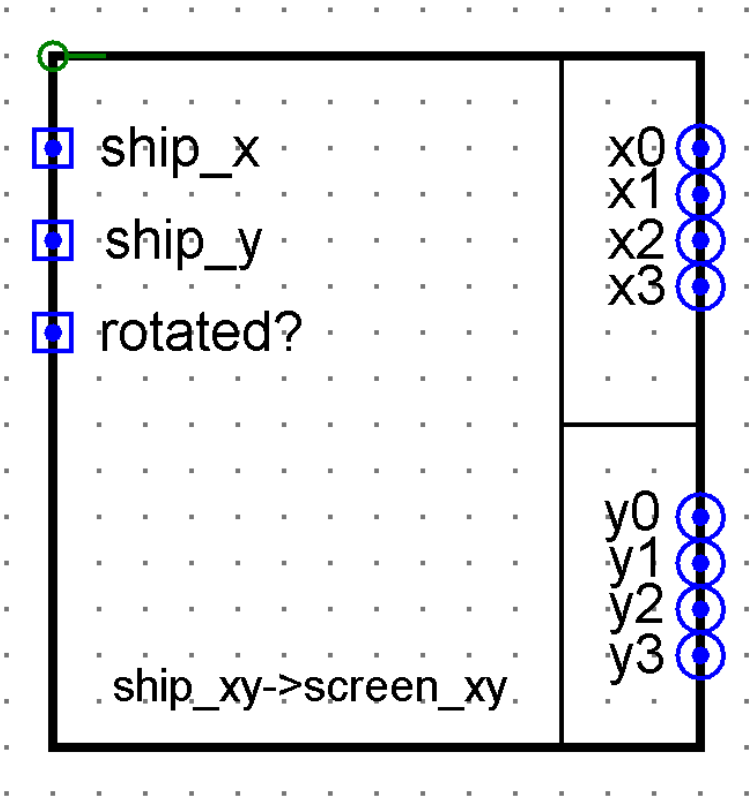


Rys. 16. Układu pozwalający na wybór danych do narysowania (look under mask)

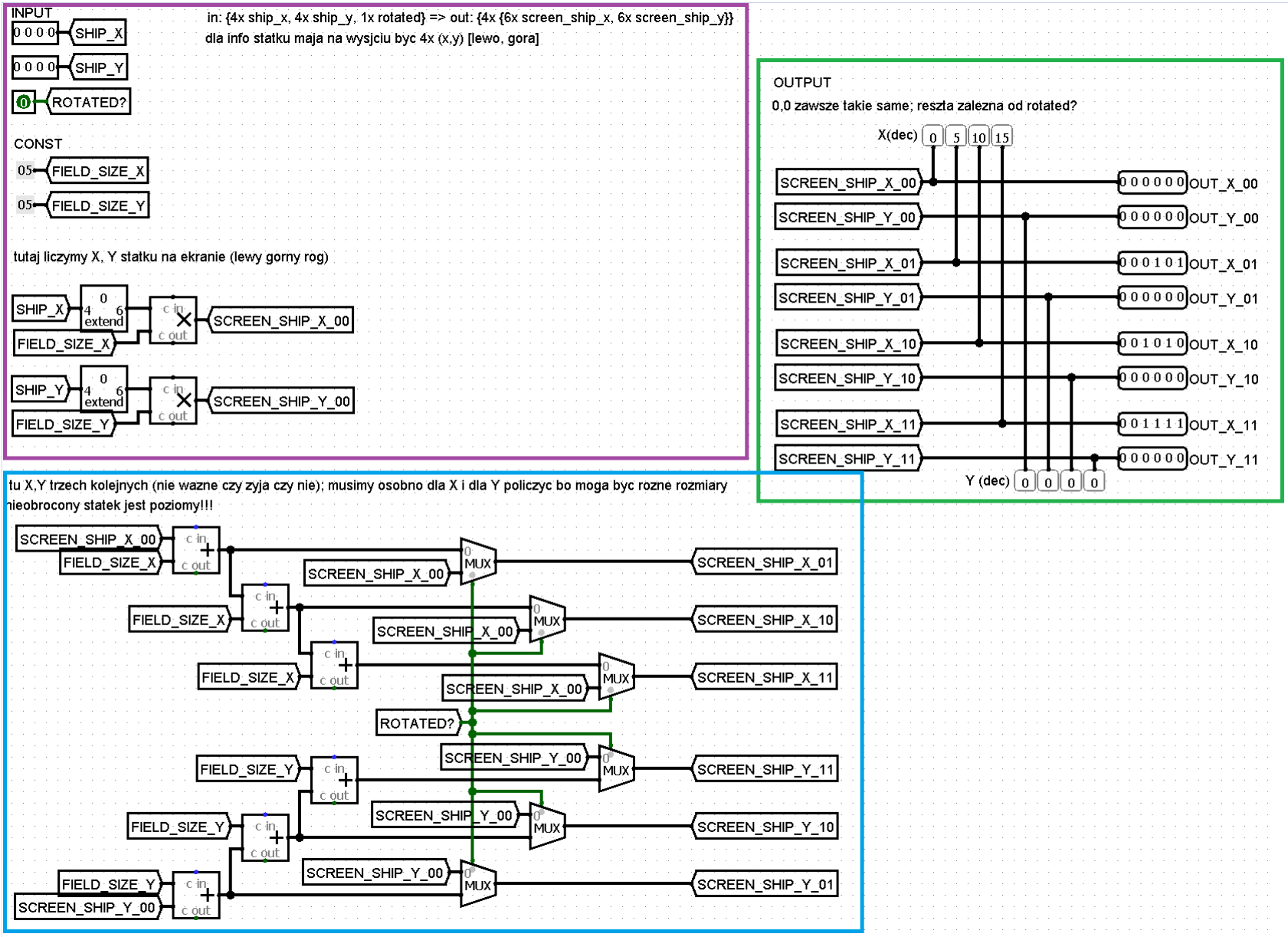
Sygnał przechodzący przez układ oznaczony kolorem pomarańczowym na Rys. 14. po przejściu przez drawing_select.circ staje się 5 bitową informacją o grafice w danej kolumnie ekranu podzielonego na partie 5 na 5. Następnie blokiem EXTEND rozszerzono liczbę bitów do 30, dzięki czemu obsłużono już pełny ekran. W aktualnej formie grafika zostałaby narysowana od dołu ekranu (środek układu współrzędnych byłby w lewym dolnym rogu), aby to obejść użyto LEFT BIT SHIFT przesuwając dane w górę. Dalej przesuwamy grafikę już o konkretną wartość określającą szerokość kursora przy pomocy RIGHT BIT SHIFT. Następnie ograniczono rysowanie grafiki na ekran jedynie do chwil w których CLOCK jest w stanie niskim, ponieważ zmienia się wtedy kolumna w drawing_select.circ. W innym wypadku powieliła by się grafika z poprzedniej kolumny. Ostatecznie sygnał wchodzi na demultiplekser, z którego poziomu wybierany jest ekran do wyświetlenia. Układ oznaczony kolorem żółtym na Rys. 14. działa analogicznie obsługując odcięte, jednakże przesunięcie następuje ręcznie – z każdym krokiem przesuwano się o jeden SCREEN_DATA do przodu (układ ponad żółtym). Licznik liczy do pięciu (od 0 do 4) bo mamy pixel 5 na 5, następnie jest to sumowane, bo przechodzimy na ekran 30 na 30. Ostateczne porównanie pozwala na wybór odpowiedniej kolumny ekranu (oś y -> wybór ekranu, oś x -> wybór kolumny ekranu wybranego przez oś y).

4.4.3. ship_xyr_to_screen_xy.circ

Układ wiążący pozycję statku w koordynatach podzielonego ekranu na 12 obszarów 5 na 5 pikseli oraz jego orientację (statek nieodwrócony jest poziomy) z jego położeniem na docelowym ekranie. Każda pamięć pojedynczego statku składa się z 4 segmentów. Oznaczono kolorem czerwonym na Rys. 10.



Rys. 17. Graficznie przygotowany układu wiążący pozycje statku wraz z rotacją oraz docelowe położenie na ekranie



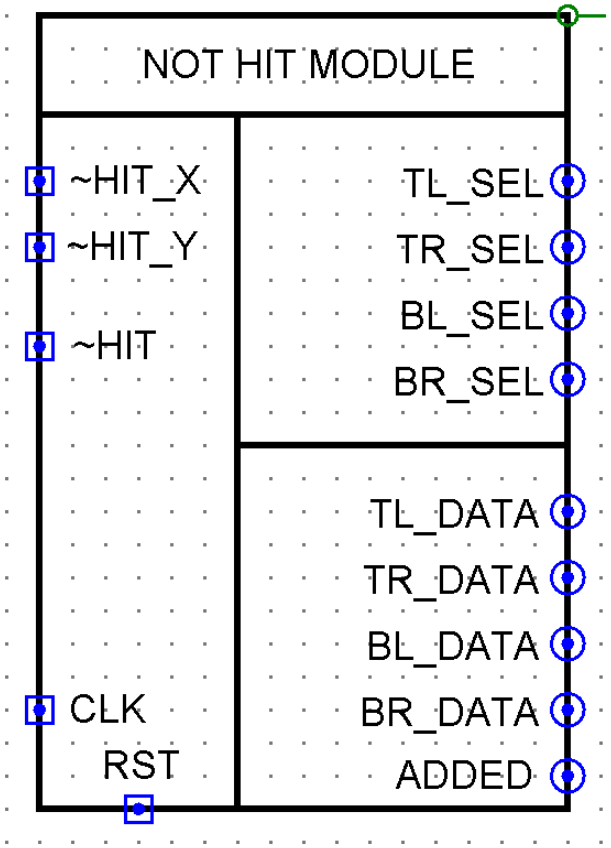
Rys 18. Układ wiążący pozycje statku wraz z rotacją oraz docelowe położenie na ekranie (look under mask)

Segment 0 obliczany jest wprost z mnożenia w fioletowym bloku Rys. 18.. Blok EXTAND pozwala na rozszerzenie ilości bitów do 6 (jest to potrzebne jedynie do przemnożenia samych wartości). Różnica w ilości bitów wynika z faktu, że do określenia wartości na podzielonym ekranie wystarczą cyfry od 0 do 11 (4 bity zapasu bo korzystamy z 4 bitów), natomiast przy rozszerzeniu do pełnego ekranu (ostateczne koordynaty od 0 do 59), potrzeba użyć już sześciu bitów. Otrzymane wartości wyświetlane są przy pomocy prostego bloku oznaczonego kolorem zielonym na Rys. 18. służy on jedynie do określeniu labeli otrzymanych wartości w celu ujednoliceniu formy bloku na Rys. 17.. Robi on dokładnie to samo, dla każdego segmentu pamięci statku. srodkowa czesc pamieci X(dec) oraz Y(dec) służy jedynie sprawdzeniu czy mnożenie zostało poprawnie wykonane.

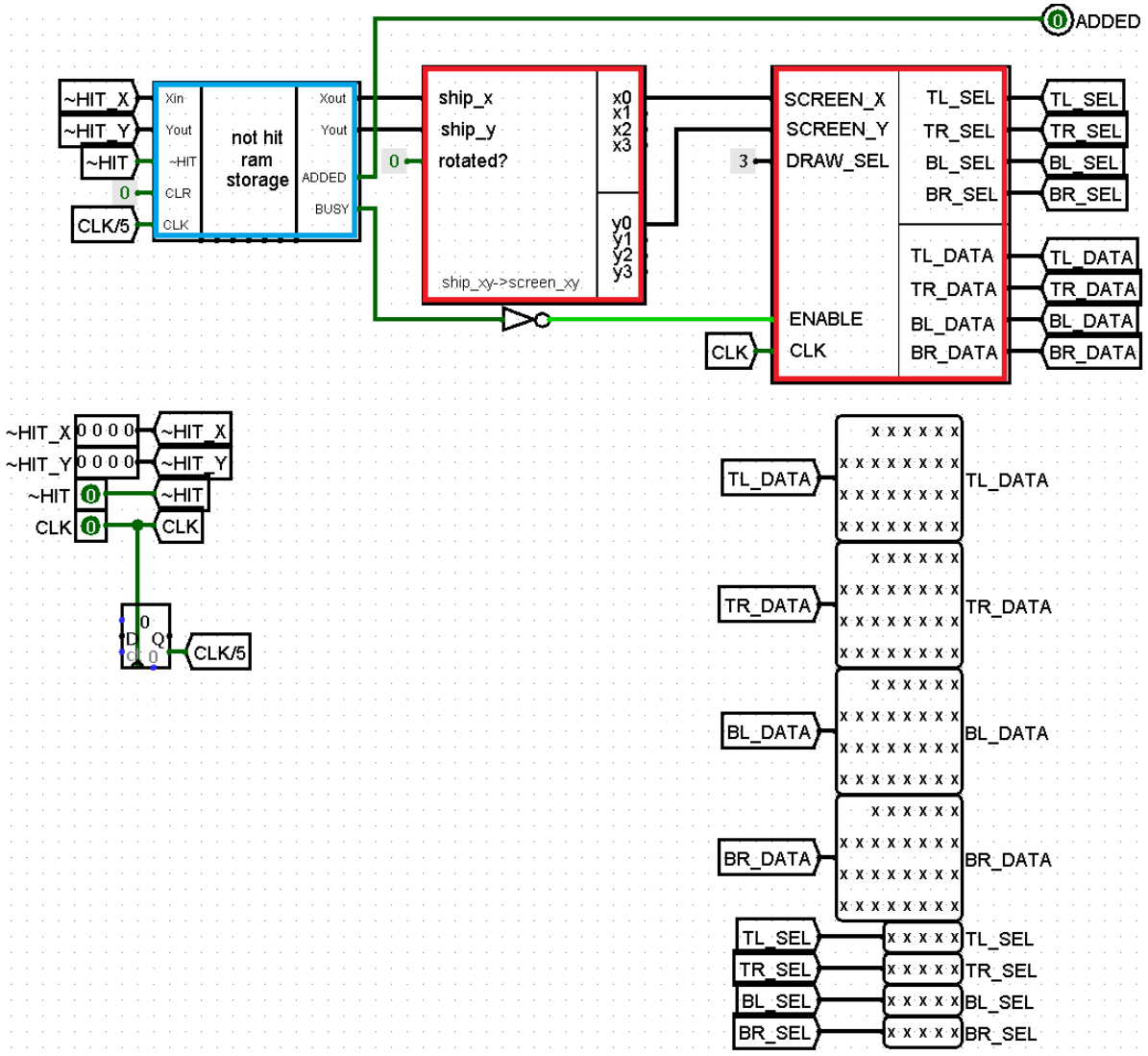
Układ oznaczony kolorem niebieskim na Rys. 18. służy do obliczenia pozostałych 3 segmentów pamięci statku. Za każdym razem (zarówno przy statku obróconym jak i nie obróconym) stała jest jedna koordynata – czy to x, czy y. Dzięki temu dodając kolejno rozmiar pola uzyskuje się pożądaną lokalizację kolejnego "piksela" statku na docelowym ekranie np. obliczając segment 1 (oznaczenia 01 na schemacie blokowym) wzięto wartość, zmiennej x lub y, z segmentu 0, a następnie dodano do niej szerokość pola. Segment 2 i 3 obliczany jest analogicznie. Aby wykonać te obliczenia użyto bloków wykonujących matematyczną sumę oraz multiplekserów, których wejście sterujące obsługuje jednobitowy sygnał obrócenia statku. Na wyjście wypisywane jest, zależnie od tego czy statek jest odwrócony, x statku lub y.

4.5. not_hit_module.circ

Układ zapewniający wyświetlanie pozycji, w których nie było uderzenia w statek. Do bloku przedstawionego na Rys. 19. wchodzi sygnał ~HIT określający czy statek został uderzony (logiczne 0 - tak; logiczne 1 - nie). Następnie ~HIT_X jest współrzędną x pozycji, w której był strzał, analogicznie ~HIT_Y - współrzędna y pozycji, w której padł strzał. Natomiast na wyjściu: pin "added" sygnalizujący o tym, że nowa lokalizacja została zapisana do pamięci, a pozostałe to piny odpowiadające za sterowanie ekranami.



Rys. 19. Graficznie przygotowany układ not_hit_module.circ

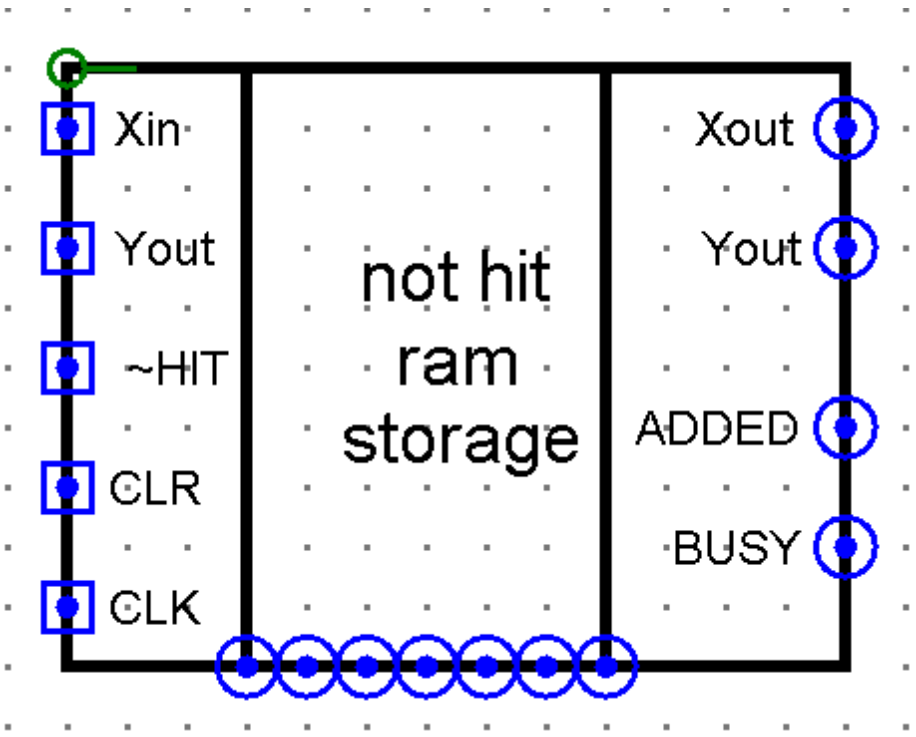


Rys. 20. Układ not_hit_module.circ look under mask

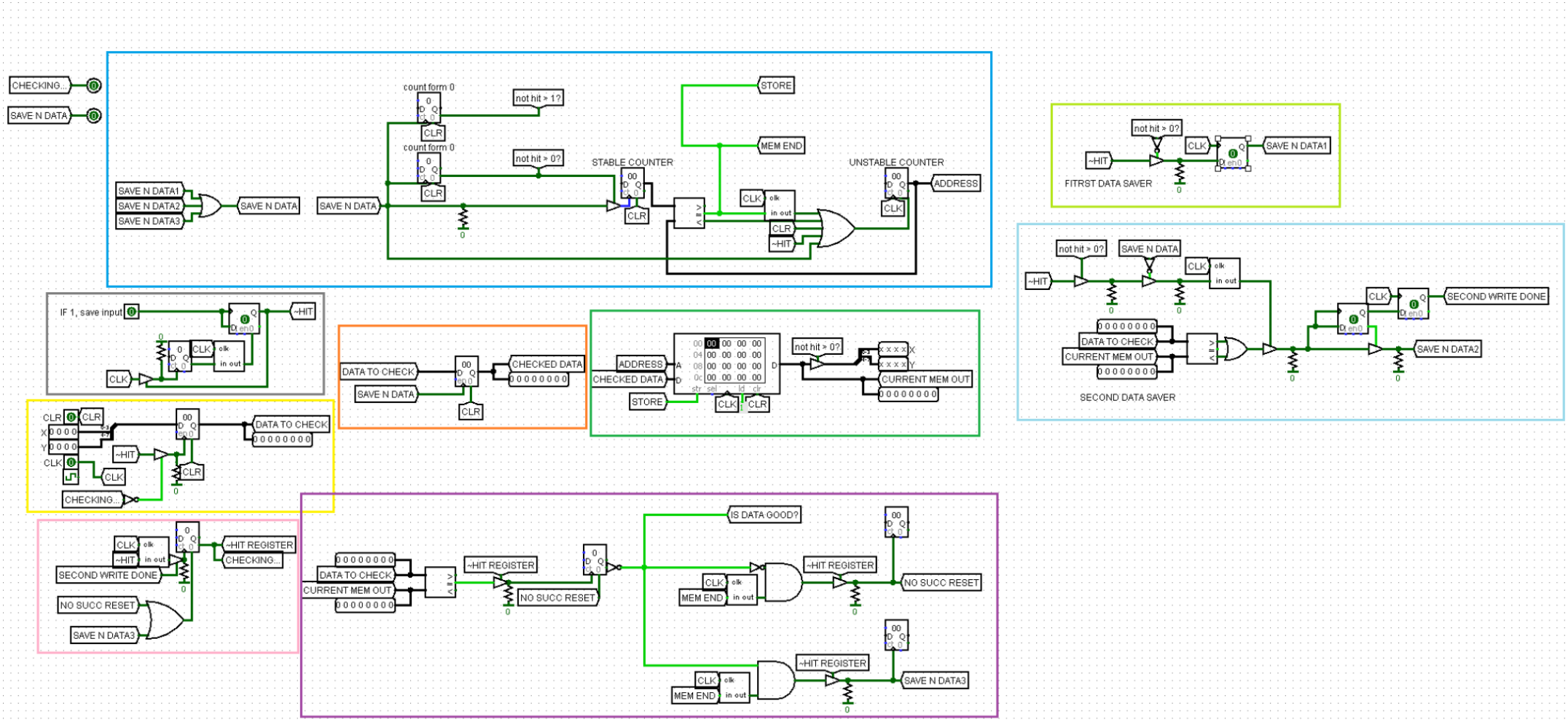
Oba układy zaznaczone na czerwono na Rys. 20. zostały opisane wcześniej są to: screen_xy_to_screen_data.circ (Rys. 13.) oraz ship_xyr_to_screen_xy.circ (Rys. 17.). Jedynie do pinu DRAW_SEL podane jest 3, ponieważ ten numer oznacza rysowanie kropki jako nietrafienia (jedynie interpretacja graficzna). Blok oznaczony kolorem niebieskim to not_hit_ram_storage.circ. Pozostałe układy poniżej obsługują wyjścia do opisania bloku końcowego.

4.5.1. not_hit_ram_storage.circ

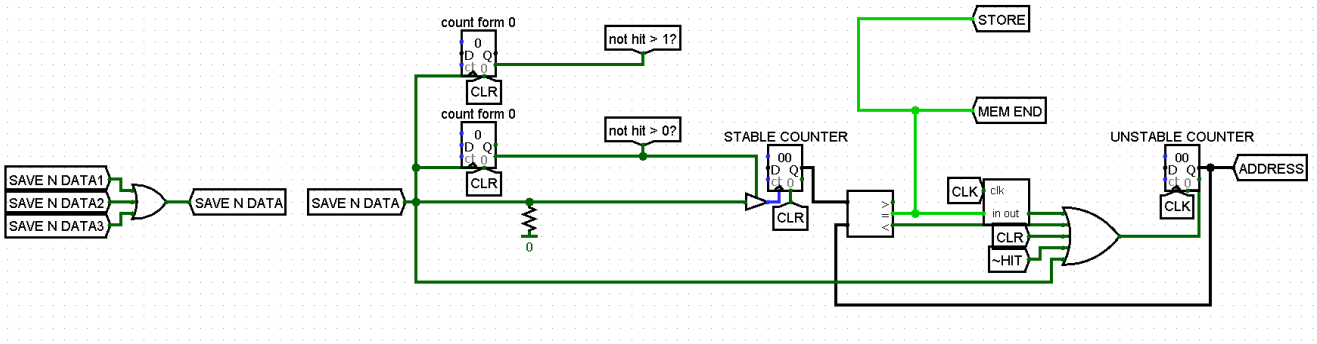
Układ odpowiadający za zapisywanie wszystkich pozycji, w których nie było trafienia i otrzymanie na wyjściu co kolejny tick zegara pozycji x oraz y, w których nie było tego trafienia. Blok przedstawiony na Rys. 21. na wejściu otrzymuje takie same sygnały jak not_hit_module.circ z Rys. 19., jedyną różnicą jest pin CLR służący do szybkiego wyczyszczenia pamięci w trakcie testów. Natomiast wyjścia: pin ADDED również jest analogiczny do opisu Rys. 19.; BUSY w stanie logicznym jeden określa to, że układ w danym momencie sprawdza czy to co jest mu podawane na wejściu znajduje się już w pamięci czy też nie, implikuje to przekazanie tej informacji na wejście układu rysującego.



Rys. 21. Graficznie przygotowany układ not_hit_ram_storage.circ

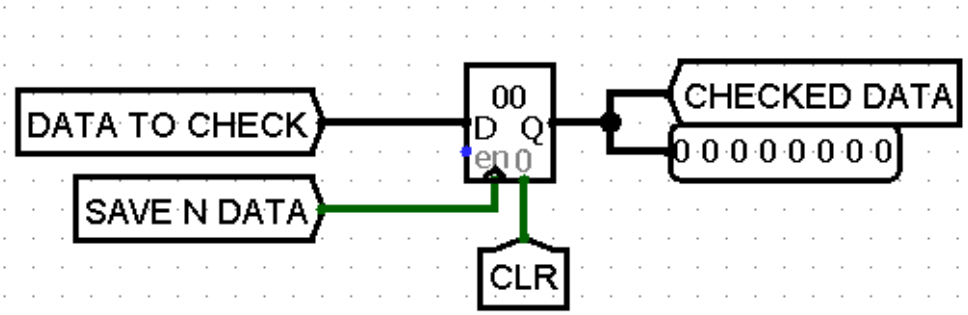


Rys. 22. Układ not_hit_ram_storage.circ (look under mask)



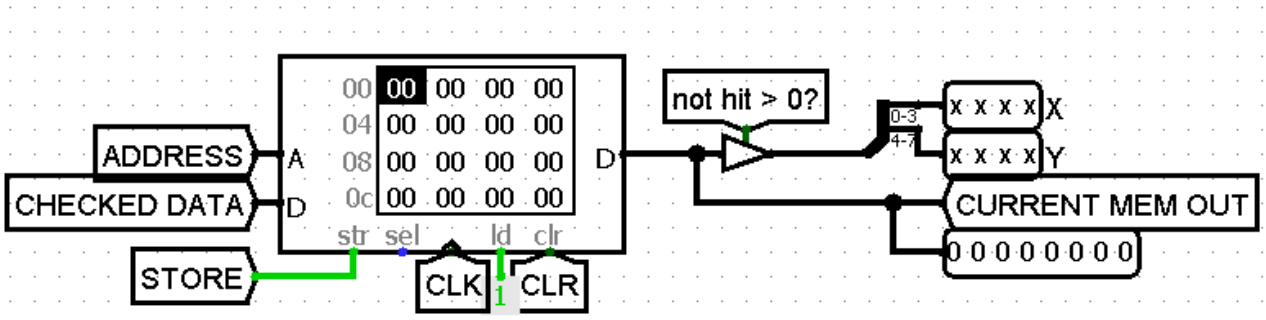
Rys. 23. Układ zaznaczony kolorem niebieskim na Rys. 22. (zbliżenie)

Układ na Rys. 23. służy do zapisywania danych do pamięci gdy stan logiczny na SAVE_N_DATA jest równy 1. Ponadto zarządza pamięcią – dodaje 1 do licznika i przechodzi na ostatnie miejsce (ostateczne miejsce zapisu w danym kroku). W przypadku gdy nie ma sygnału do zapisania w kółko zmienia adres, który jest podawany na wyjście.



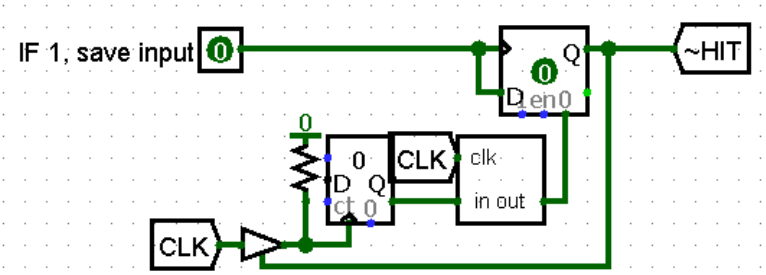
Rys. 24. Układ zaznaczony kolorem pomarańczowym na Rys. 22. (przybliżenie)

Układ na Rys. 24. sprawia, że dane na wejściu są dostępne dla pamięci jedynie w przypadku gdy następuje rozkaz zapisu.



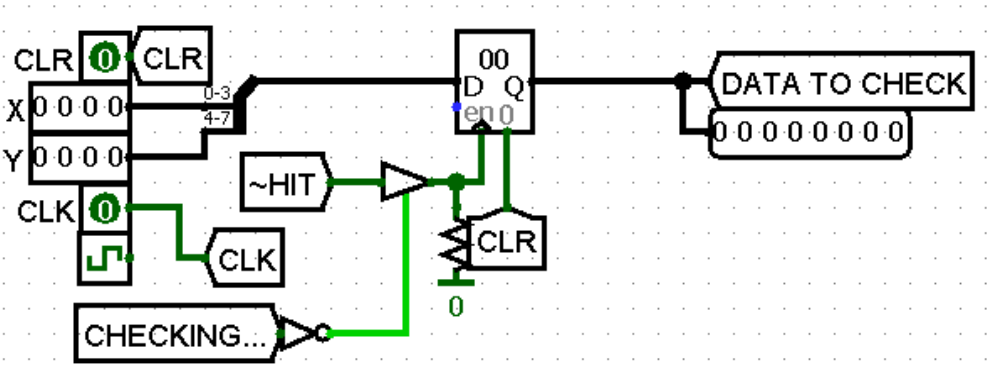
Rys. 25. Układ zaznaczony kolorem zielonym na Rys. 22. (przybliżenie)

Układ na Rys. 25. jest organizacją pamięci RAM, zarządzaną przez inne moduły. Nie wyświetla nic gdy nie było rozkazu zapisu do pamięci.



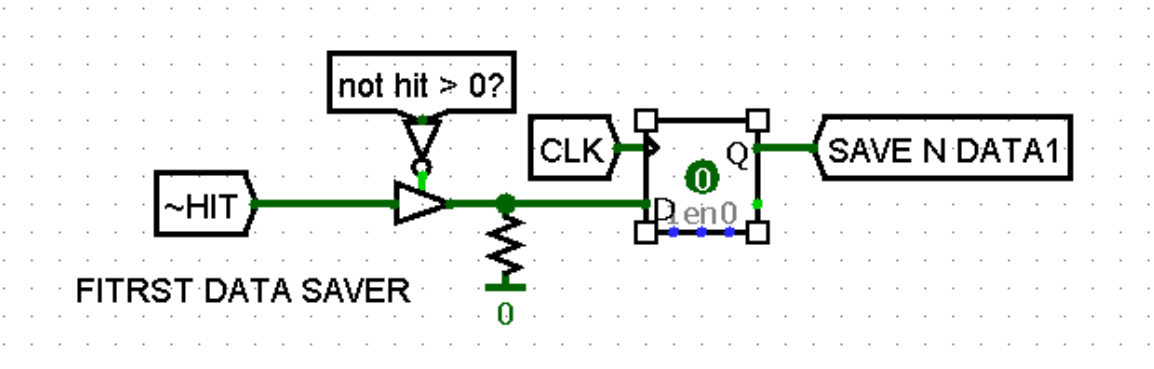
Rys. 26. Układ zaznaczony kolorem szarym na Rys. 22. (przybliżenie)

Układ na Rys. 26. zarządza zadanym sygnałem – jeśli na wejściu pojawi się 1 to trzyma stan wysoki przez 2 ticki zegara, a następnie zmienia stan wyjścia na 0.



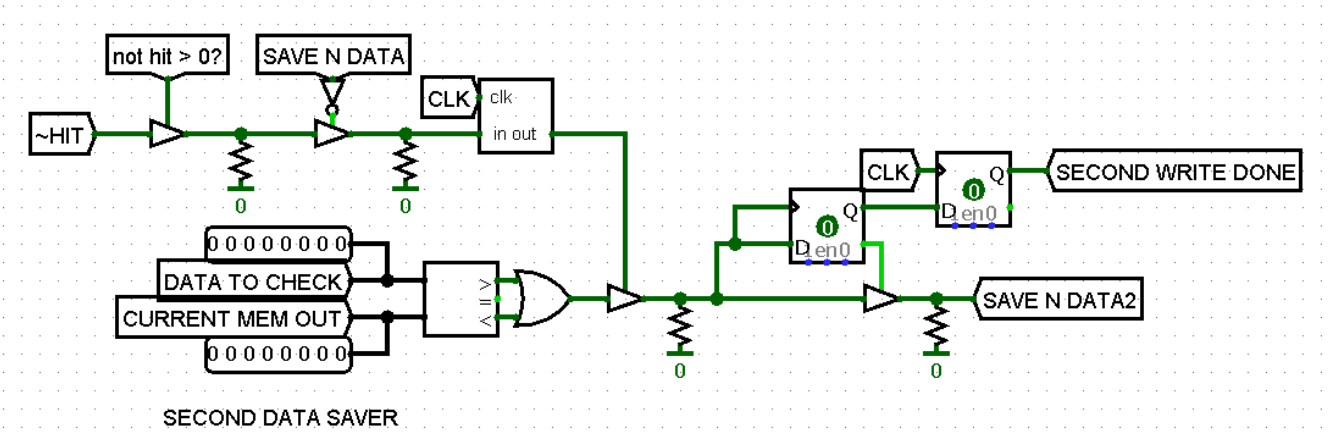
Rys. 27. Układ zaznaczony kolorem żółtym na Rys. 22. (przybliżenie)

Gdy wejście ~HIT układu na Rys. 27. jest w stanie wysokim – oznacza to, że statek został nietrafiony – układ zapisuje do rejestru pozycję statku. CHECKING… zabezpiecza przed rozkazem zapisu do pamięci w przypadku, gdy aktualnie odbywa się sprawdzanie (flaga busy).



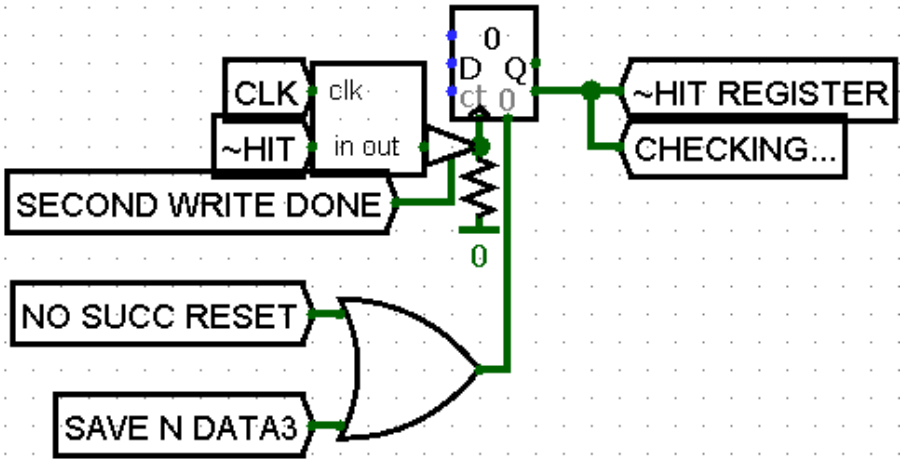
Rys. 28. Układ zaznaczony kolorem jasnozielonym na Rys. 22. (przybliżenie)

Układ na Rys. 28. odpowiada za zapis na pozycję numer jeden, bez wykonywania wcześniej jakiegokolwiek sprawdzania.



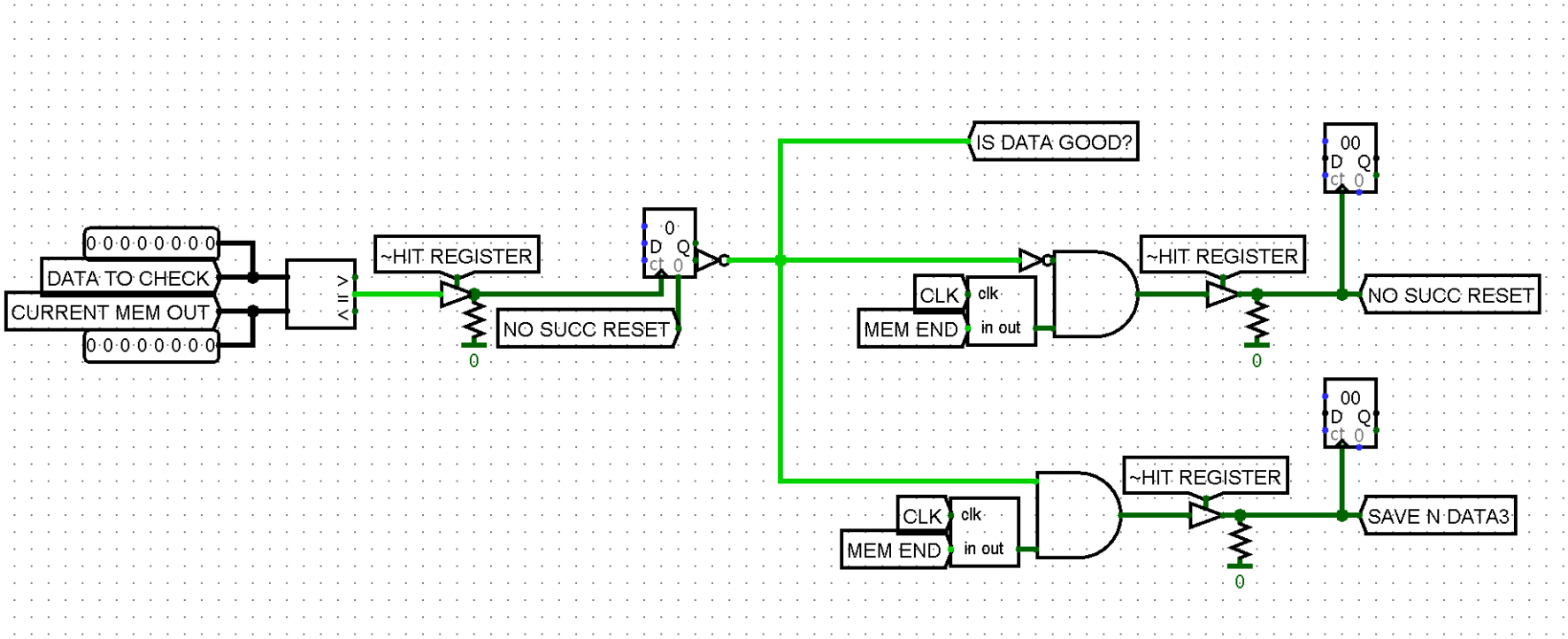
Rys. 29. Układ zaznaczony kolorem jasnoniebieskim na Rys. 22. (przybliżenie)

Układ na Rys. 29. odpowiada za zapisanie na drugie miejsce w pamięci z wykonaniem sprawdzenia czy nie ma już tego na pierwszym miejscu w pamięci. Dopiero po wykonaniu drugiego zapisu działają układy z rysunków 30. oraz 31..



Rys. 30. Układ zaznaczony kolorem różowym na Rys. 22. (przybliżenie)

Układ na Rys. 30. trzyma stan 1 tak długo aż nie zostanie wykonane sprawdzenia lub zapisanie stanu.

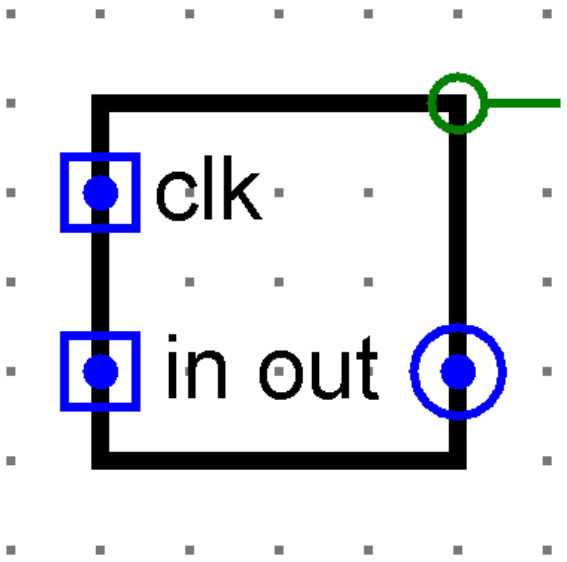


Rys. 31. Układ zaznaczony kolorem fioletowym na Rys. 22. (przybliżenie)

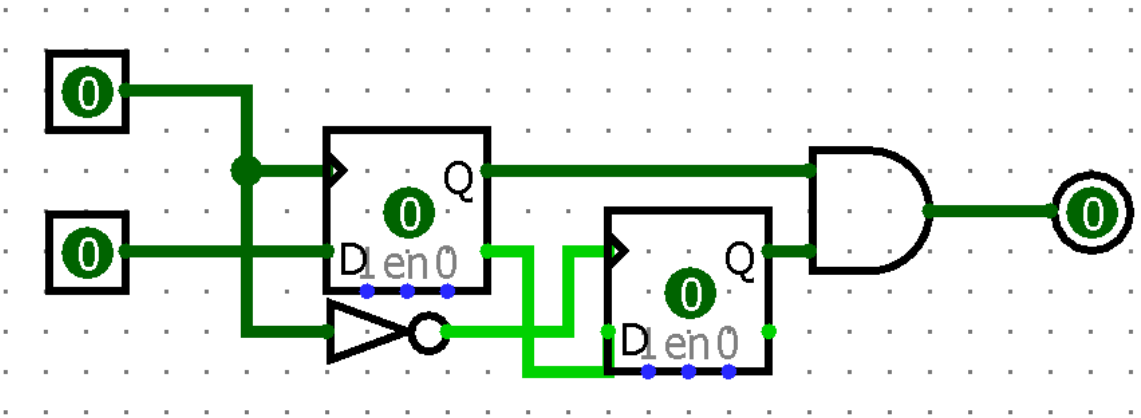
Układ na Rys. 31. jest główną logiką, która porównuje wszystkie dane na wejściu z tym co jest w pamięci. Jeżeli po przejściu wszystkich miejsc w pamięci okazuje się, że dana lokalizacja jest już wcześniej zapisana w pamięci to robi reset na układzie z Rys. 30.. Jeżeli natomiast okazuje się, że danej pozycji nie ma w pamięci następuje do niej zapis. Po dokonanym zapisie układ z Rys. 30. również jest resetowany.

4.6. tick.circ

Układ wysyła jeden raz na tick zegara dla stanu jeden na wejściu IN – jeżeli na wejściu IN jest stan wysoki to na wyjściu OUT pojawi się jeden słupek zegara.



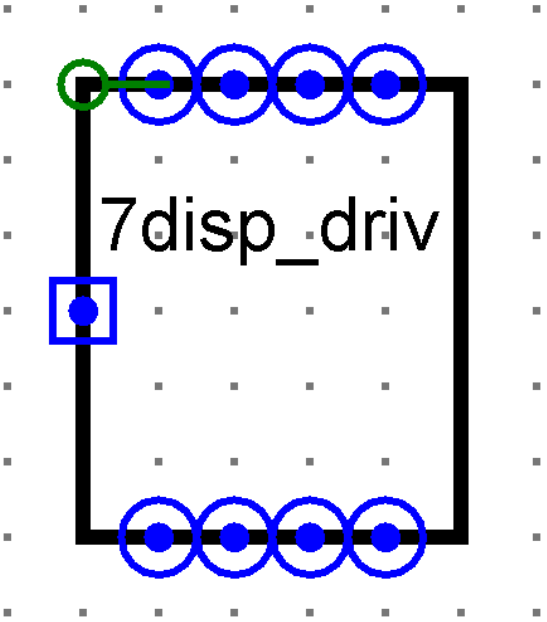
Rys. 32. Układ tick.circ schemat bloku



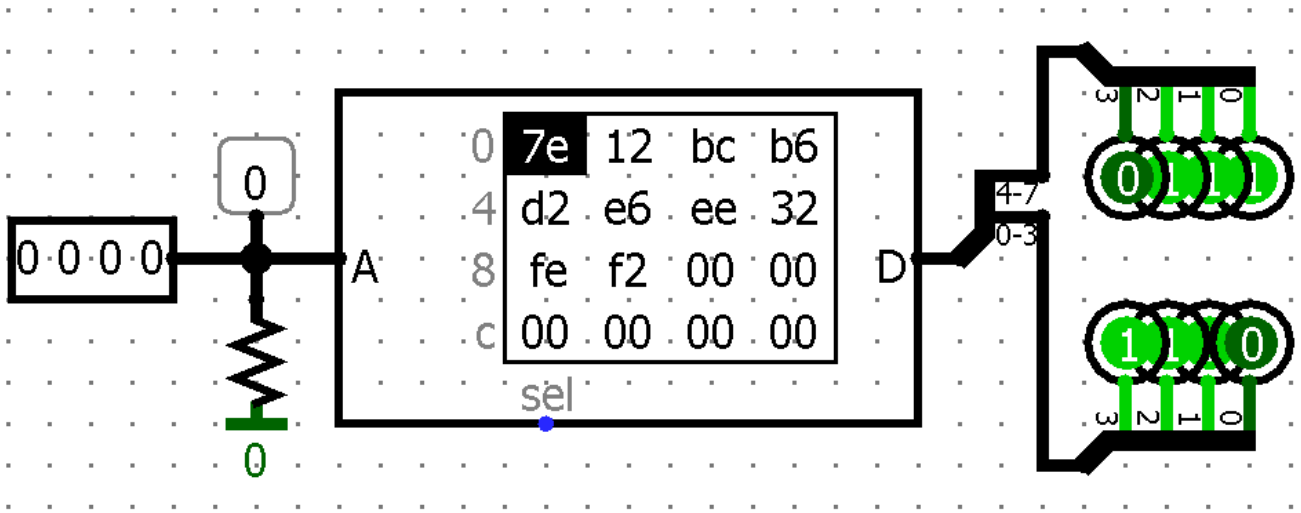
Rys. 33. Układ tick.circ (look under mask)

4.7. 7seg_disp_driver.circ

Układ obsługujący wyświetlanie liczb. Zamienia liczbę zapisaną na 4 bitach na graficzne wyświetlenie jej w systemie dziesiętnym. Układ na wejściu otrzymuje liczbę zapisaną na 4 bitach, ten sygnał obsługuje pamięć ROM, z której wybierana jest wartość zapisana na 8 bitach. Informacja zostaje przekazana na wyjście docelowo wchodzące na wyświetlacz siedmiosegmentowy. Układ działa jedynie dla cyfr od 0 do 9. W przypadku liczby większej nic nie jest wyświetlane.



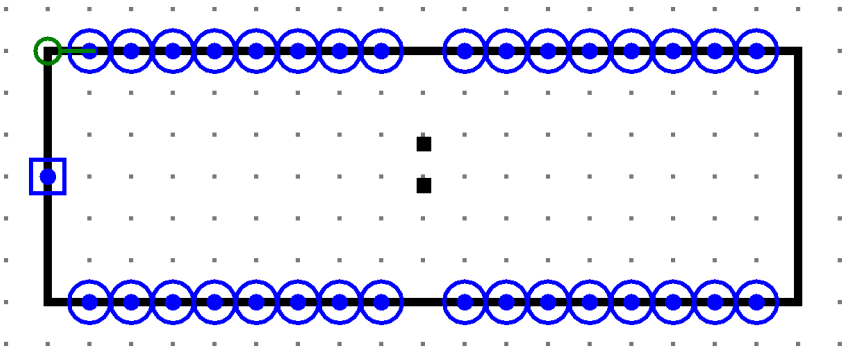
Rys. 34. Układ 7seg_disp_driver.circ schemat bloku



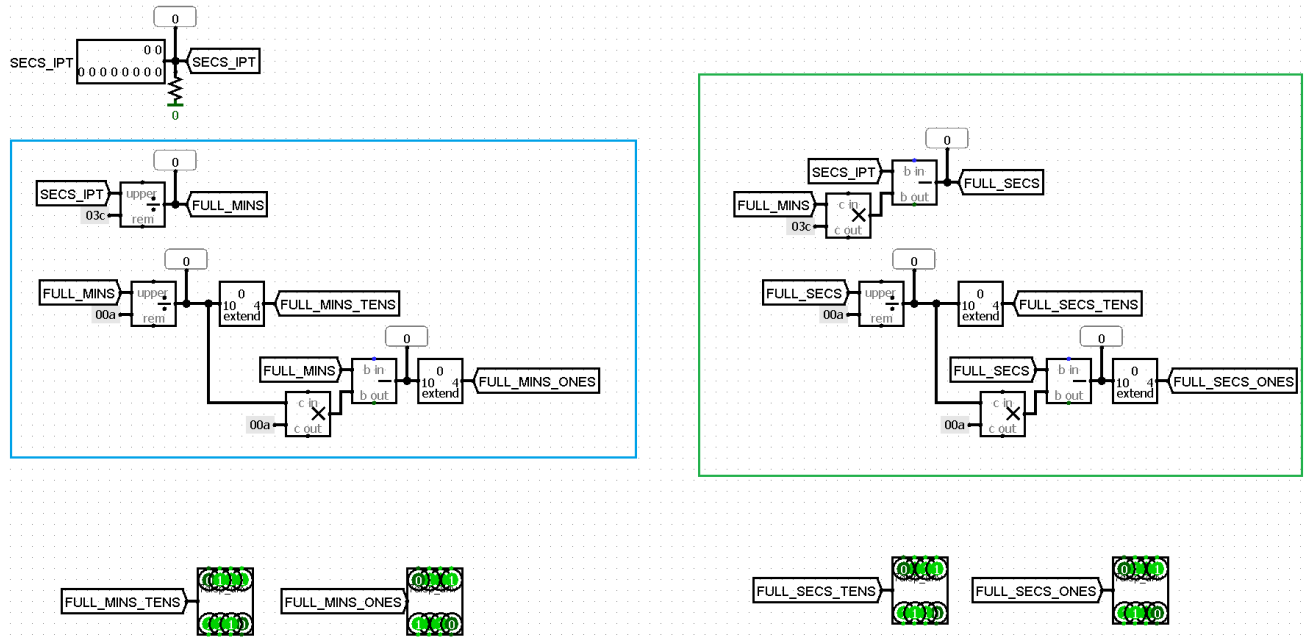
Rys. 35. Układ 7seg_disp_driver.circ (look under mask)

4.8. secs_to_score.circ

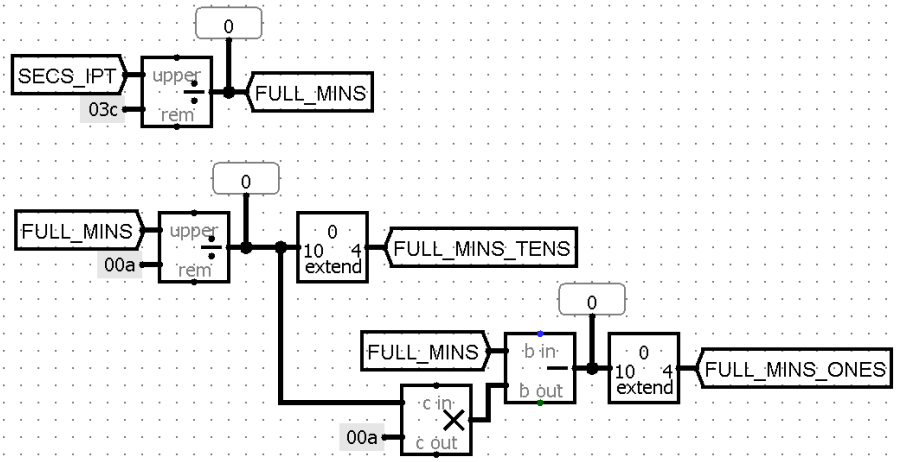
Układ zamieniający dziesięcio bitowe wejście wyrażane w sekundach na minuty oraz sekundy. Maksymalny czas rozgrywki wynosi 2^{10} sekund $\simeq$ 17 minut (do tej wartości zegar liczy poprawnie). Otrzymane na wejściu sekundy zostają podzielone przez 60 w celu otrzymania pełnych minut (dzielenie całkowite). Logika układu pozwala na wyjściu otrzymać 4 wyświetlacze siedmiosegmentowe wyświetlające minuty i sekundy upływającego czasu rozgrywki.



Rys. 36. Układ secs_to_score.circ schemat bloku

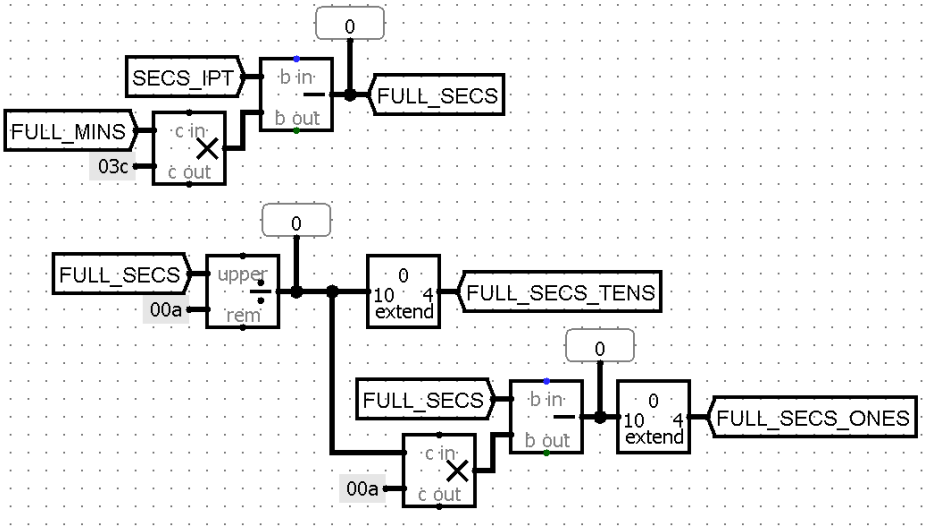


Rys. 37. Układ secs_to_score.circ (look under mask)



Rys. 38. Układ zaznaczony kolorem niebieskim na Rys. 37. (przybliżenie)

Układ na Rys. 38. pozwala na “wyciągnięcie” z dzielenia całkowitego reszty ułamka. Uzyskaną uprzednio wartość (po dzieleniu przez 60) podzielono, tym razem przez 10, dzięki temu otrzymano pełne minuty. Następnie te wartosc pomnożoną przez 10 odjęto od pełnych minut i uzyskano dzięki temu osobno jedności i dziesiątki minut

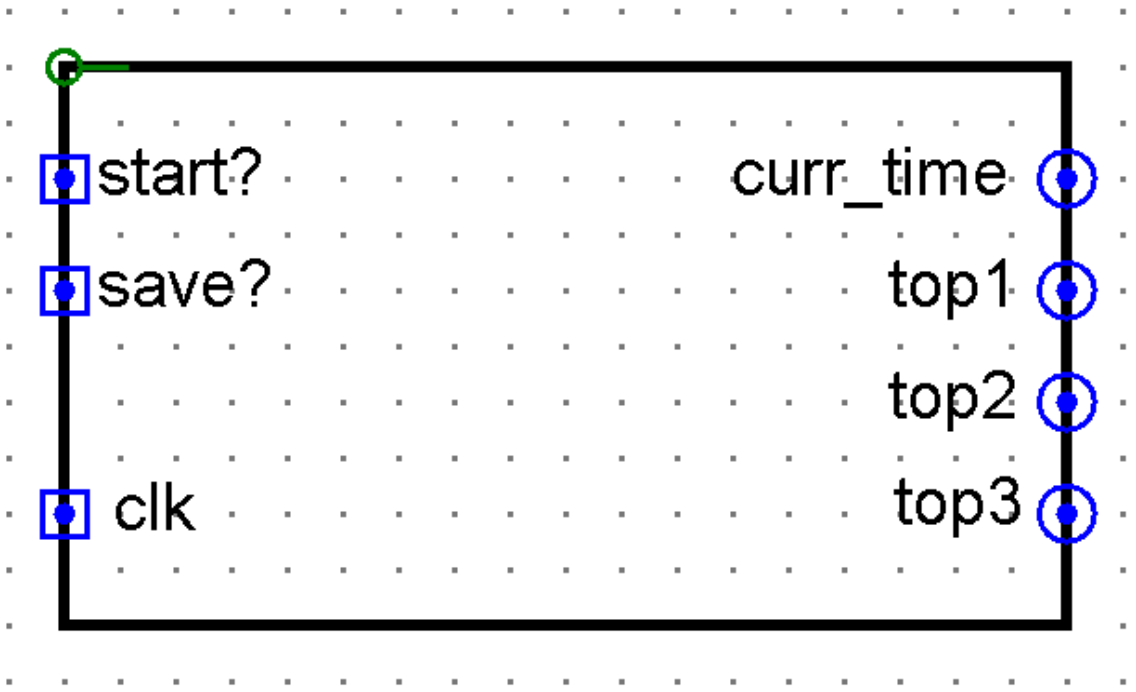


Rys. 39. Układ zaznaczony kolorem zielonym na Rys. 37. (przybliżenie)

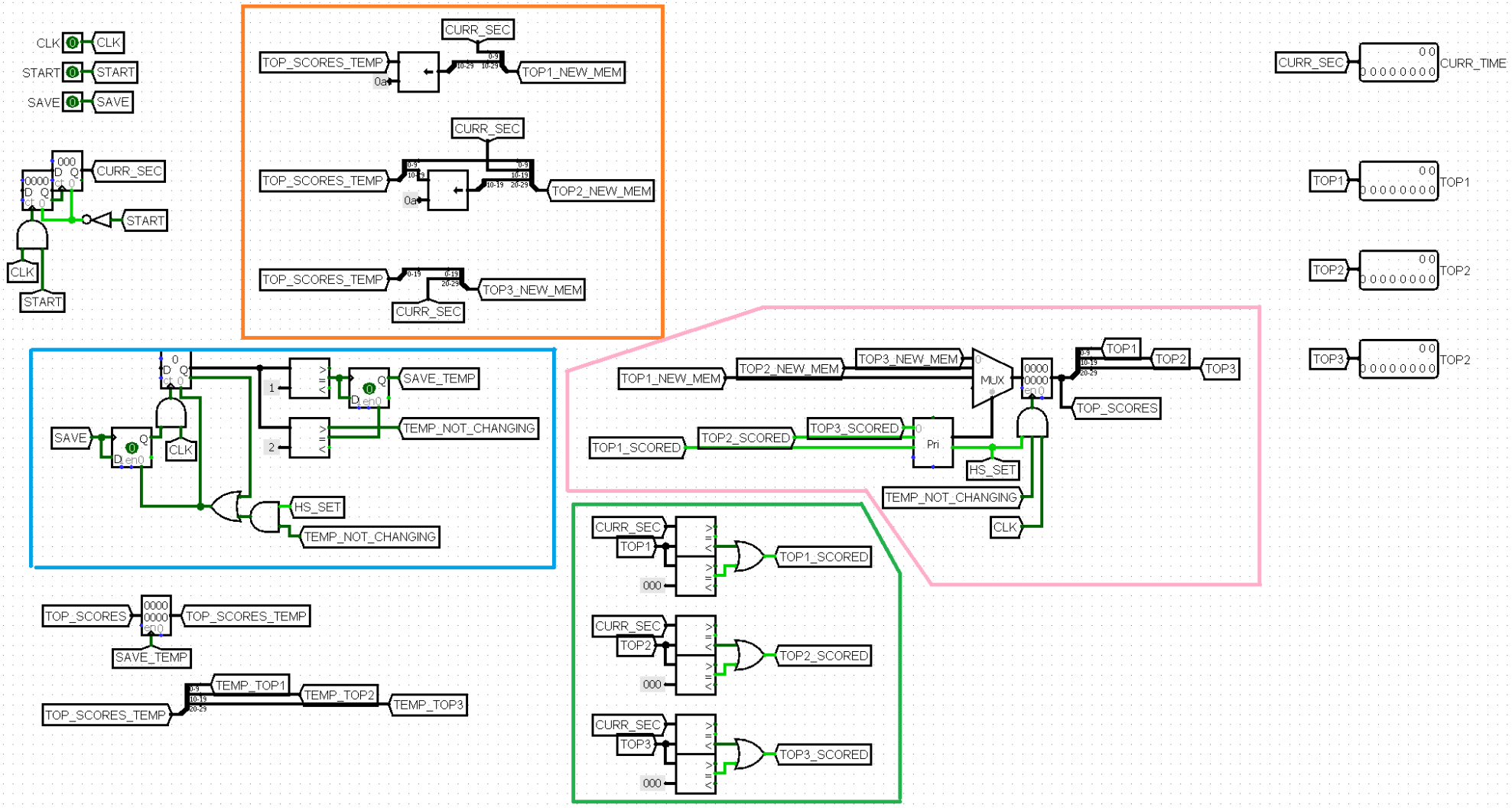
Układ przedstawiony na Rys. 39. jest odpowiednikiem układu z Rys. 38. jednak dotyczy on jedynie pełnych sekund i ich dziesiątych części. Uprzednio policzony minuty są mnożone razy 60 i odejmowane są od nich pełne sekundy. Układ poniżej jest kopią logiczną Rys. 38..

4.9. timer.circ

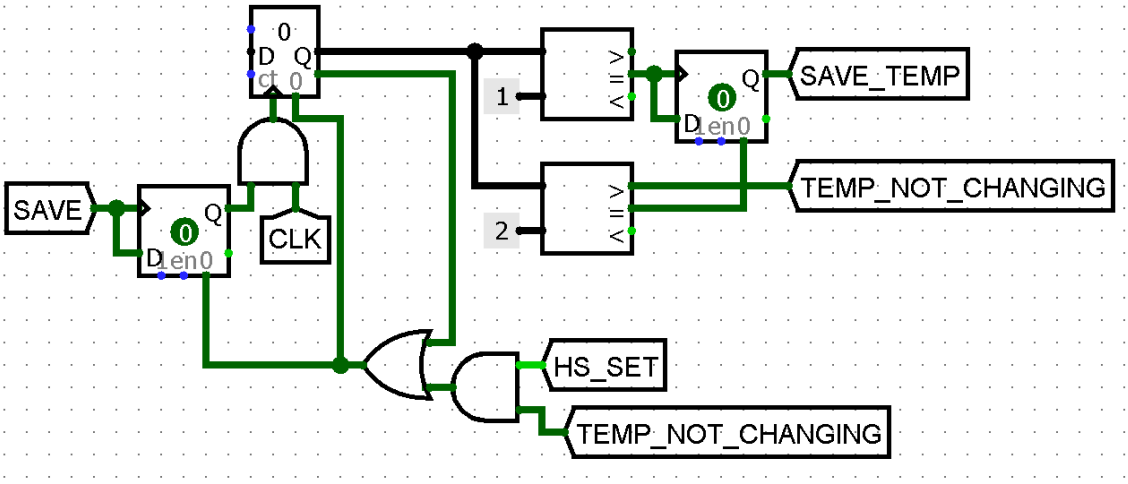
Zadaniem układu jest wyświetlenie aktualnej tabeli najwyższych wyników w historii gry oraz obsługa nadpisywania ich na podstawie aktualnych rekordów. Na wejściu otrzymuje sygnał zegarowy, START oraz SAVE. Przy utrzymaniu stanu wysokiego na wejściu START oraz zmieniającym się sygnale zegarowym timer.circ zaczyna naliczać czas rozgrywki. Wejście SAVE obsługuje zapis do wyświetlenia najwyższych wyników.



Rys. 40. Układ timer.circ schemat bloku

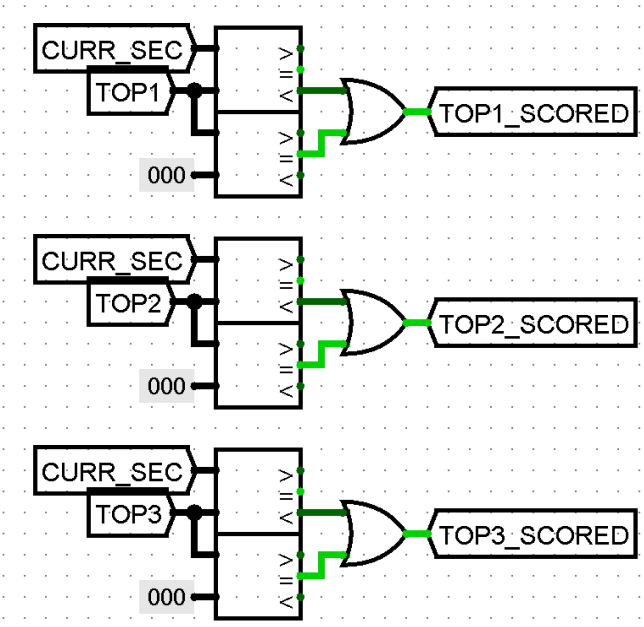


Rys. 41. Układ timer.circ (look under mask)



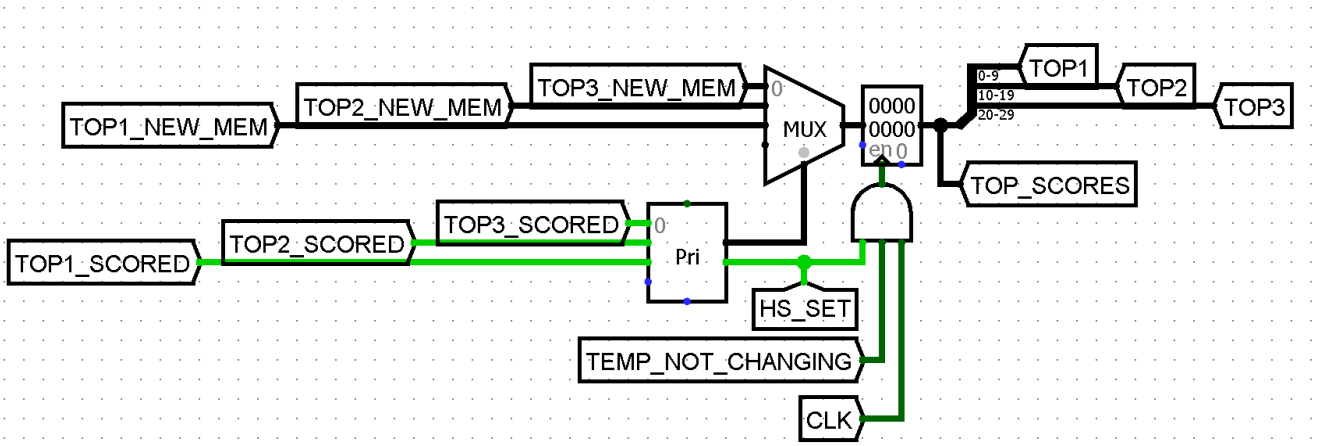
Rys. 42. Układ zaznaczony kolorem niebieskim na Rys. 41. (przybliżenie)

Układ z Rys. 42. służy do zapisywania aktualnego wyniku, najpierw sprawdza sekundowo czy dany wynik jest większy od top 1, top 2, top 3. Na wyjściu przekazuje wartość logiczną określającą czy dany wynik ma zostać zapisany (logiczne 1), czy też nie (logiczne 0).



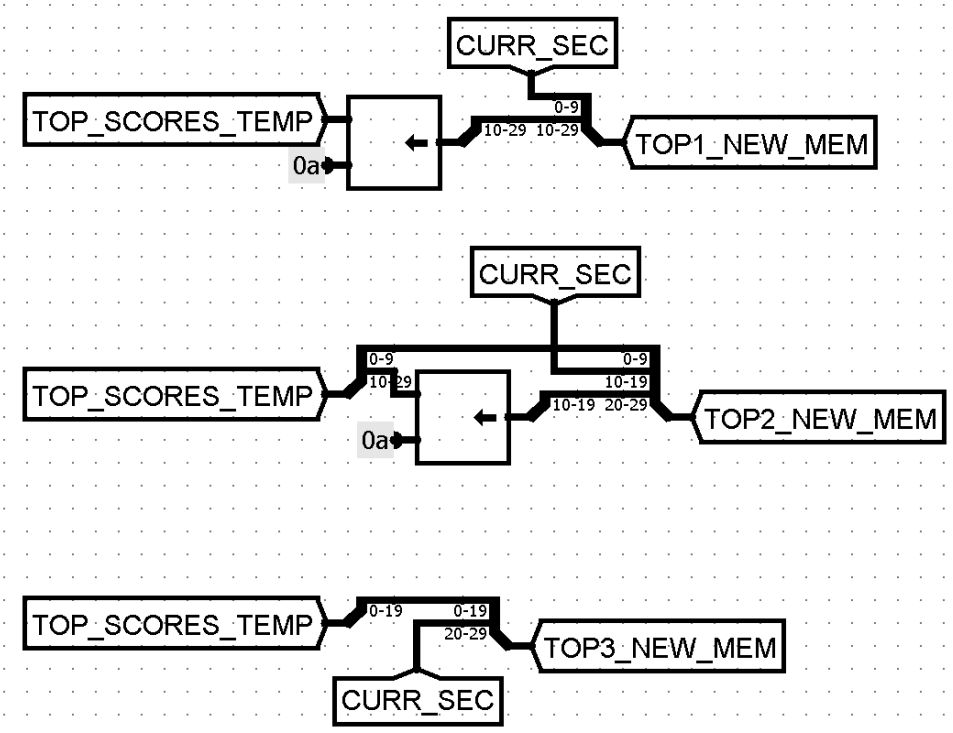
Rys. 43. Układ zaznaczony kolorem zielonym na Rys. 41. (przybliżenie)

Część układu z Rys. 41. przedstawiona na Rys. 43. porównuje aktualny wynik z wynikami najlepszymi, a następnie jeśli spełni warunek logiczny mniejszości przechodzi na flagę np. TOP1_SCORED. Wartości równolegle przyrównywane są również do zera w celu obsługi wartości początkowej.



Rys. 44. Układ zaznaczony kolorem różowym na Rys. 41. (przybliżenie)

Otrzymane na wyjściu układu z Rys. 43. flagi używane są w układzie z Rys. 44.. Są one na wejściu PRIOR ENCODERA (przy samych stanach wysokich na wejściu przypisuje indeksy rosnąco, w tym przypadku TOP1_SCORED otrzyma indeks 0, TOP2_SCORED otrzyma indeks 2, a TOP3_SCORED otrzyma indeks 2), po którego opuszczeniu obsługują multiplekser wybierający pamięć do zapisu w tabeli wyników.

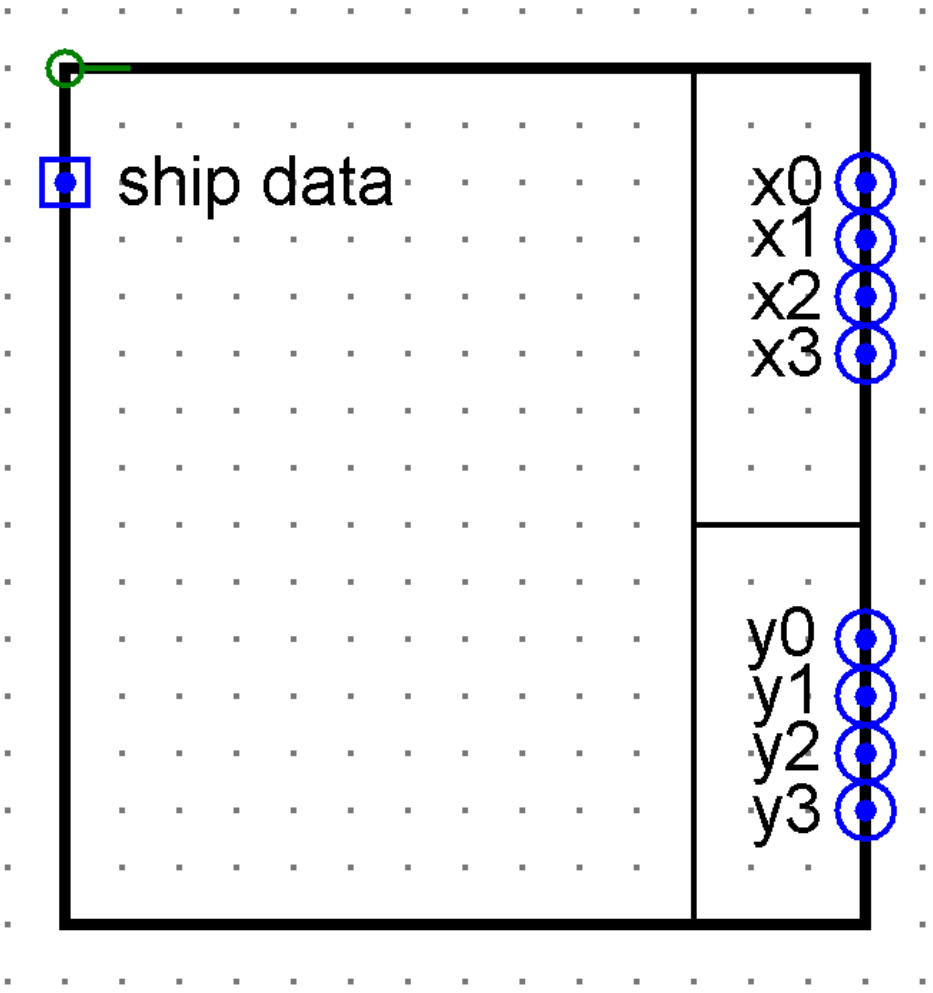


Rys. 45. Układ zaznaczony kolorem pomarańczowym na Rys. 41. (przybliżenie)

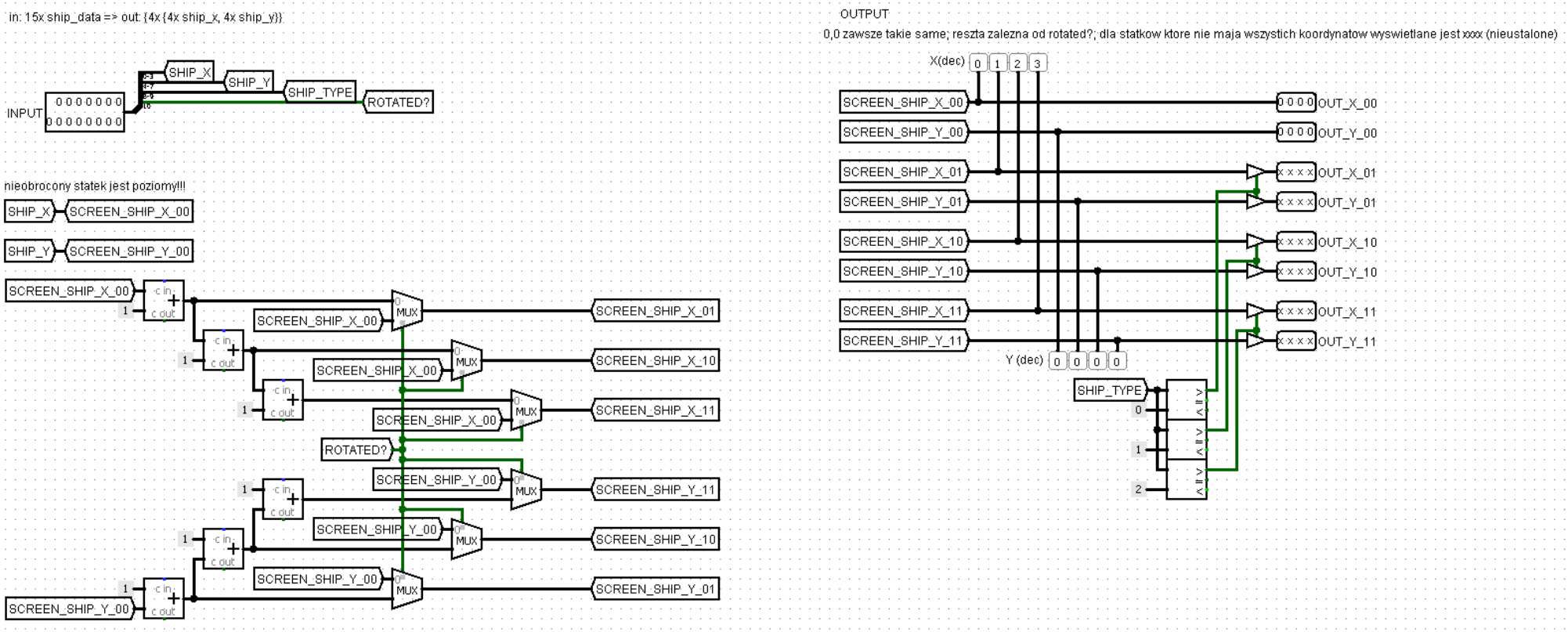
Układ z Rys. 45. przesuwa odpowiednie miejsca w końcowej tabeli wyników. Gdy nowa wartość otrzyma miejsce nr 1, poprzednie miejsca nr 1 oraz nr 2 zostają przesunięte o jedno w dół. Gdy nowa wartość osiągnie miejsce nr 2, poprzednie miejsca nr 2 i 3 zostają przesunięte. Gdy nowa wartość zajmuje miejsce nr 3, stare miejsce trzecie zostaje jedynie nadpisane nowym. Otrzymano to dzięki pracy na 30 bitach (po 10 na każde z miejsc), użyto przesunięcia bitowe w lewo. Otrzymana nowa pamięć zostaje przekazana do bloku z Rys. 44.

4.10. ship_data_to_all_ship_xy.circ

Układ jest skonstuowany analogicznie do układu ship_xyr_to_screen_xy.circ ze strony czternastej jednak jego przeznaczenie jest inne oraz rozmiar pola jest równy 1 na 1, a w tamtym było 5 na 5. Podając zapisaną pamięć z RAMu na wejście układu zamieni on ją na wyjściu na wszystkie cztery pozycje statków (współrzędną x oraz y pierwszego piksela, czy jest odwrócony, czy nie). Podstawową funkcjonalnością układu jest to czy dany statek wyjdzie za ekran czy też nie. Jeżeli statek jest typu np. typu 1, to na pozostałych wyjściach otrzyma się stan nieustalony.



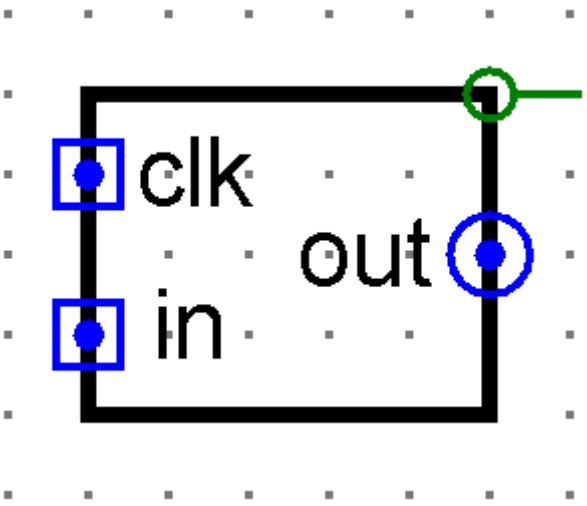
Rys. 46. Graficzne przedstawienie schematu układu ship_data_to_all_ship.circ (look under mask)



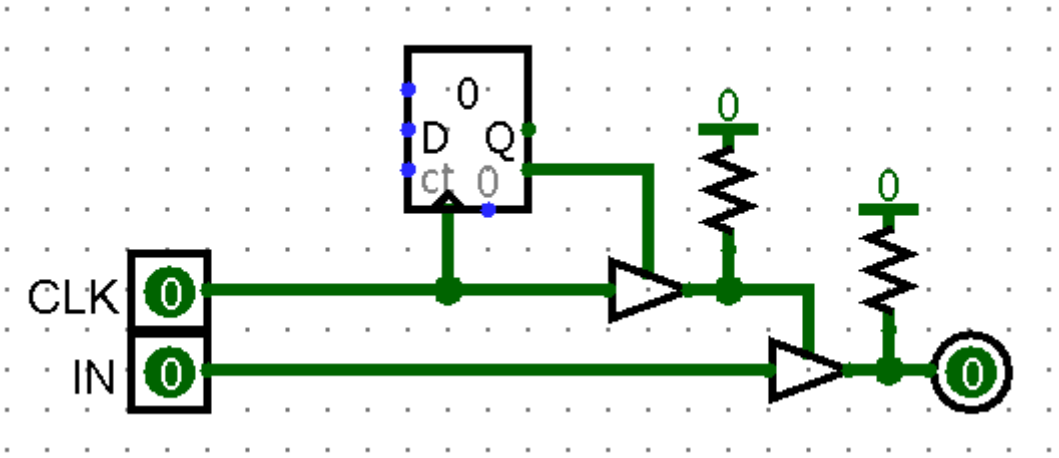
Rys. 47. Układ ship_data_to_all_ship.circ (look under mask)

4.11. tick2.circ

Układ wysyła dwa razy na tick zegara dla stanu jeden na wejściu IN – jeżeli na wejściu IN jest stan wysoki to na wyjściu OUT pojawią się dwa słupki zegara.



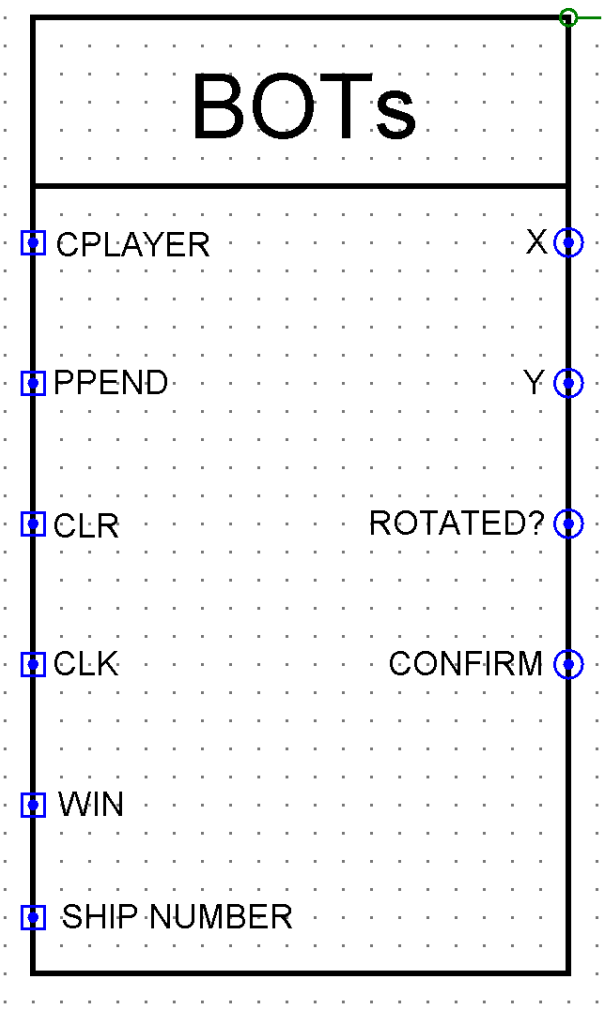
Rys. 48. Układ tick2.circ schemat bloku



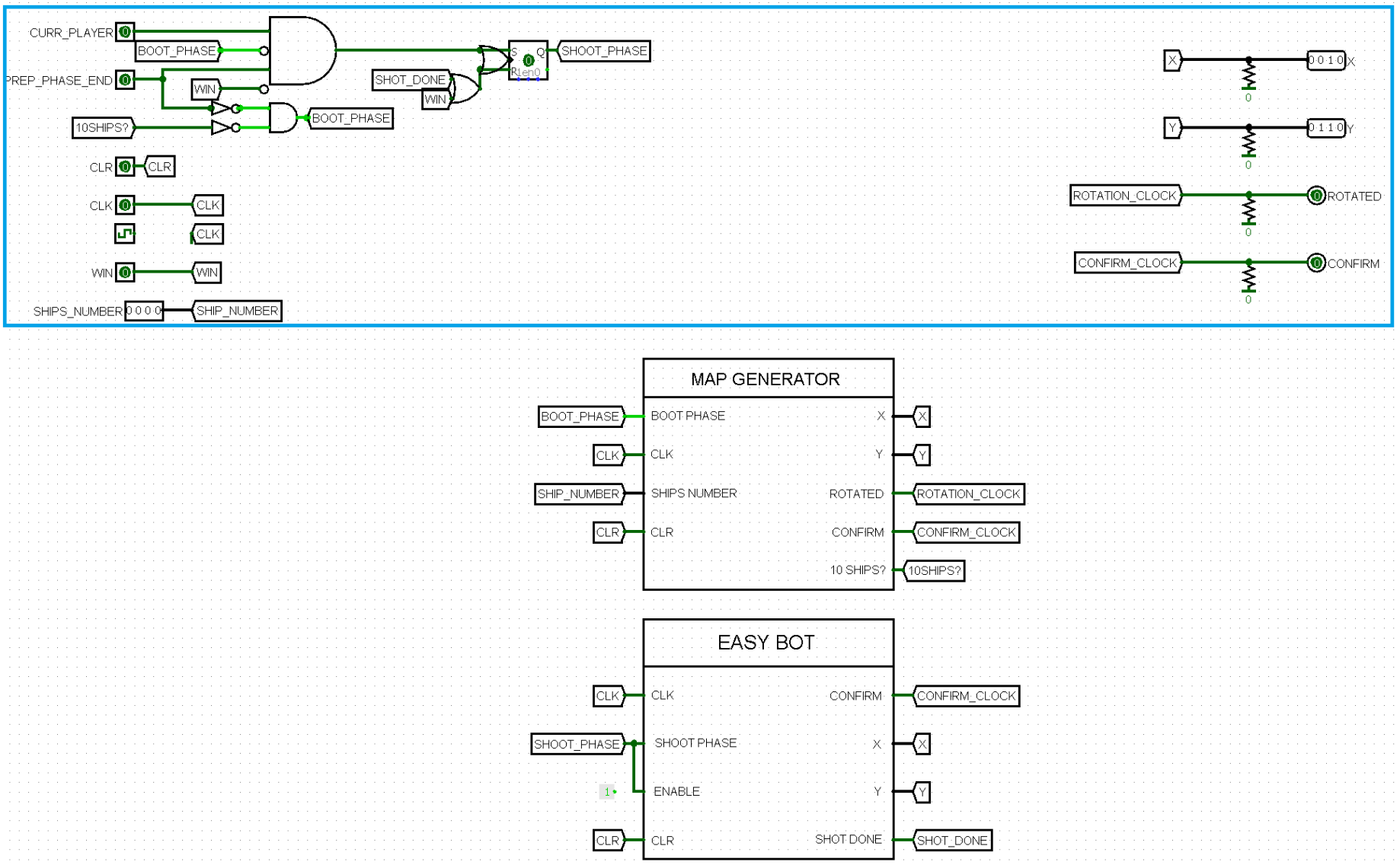
Rys. 49. Układ tick2.circ (look under mask)

4.12. bots.circ

Zadaniem układu jest obsługa bota grającego z osobą rzeczywistą. Na wejście bloku układu, przedstawionego na Rys. 50., otrzymuje sygnał CPLAYER określający czy aktualną turą jest tura bota, PPED określające czy zakończyła się faza generowania mapy – start gry, CLR pozwalające na zewnętrzny reset, sygnał zegarowy CLK, sygnał WIN odpowiedzialny za to czy gra już się zakończyła (czy ktoś już wygrał) oraz SHIP NUMBER, który jest informacją ile statków jest aktualnie na planszy. Wyjściami tego bloku są: x – koordynata osi OX, w którą bot ma strzelić, y – koordynata osi OY, w którą bot ma strzelić, ROTATED? – czy statek ma być obrócony (działa jedynie w momencie generowania mapy), CONFIRM – potwierdzenie strzału.



Rys. 50. Układ bots.circ schemat bloku

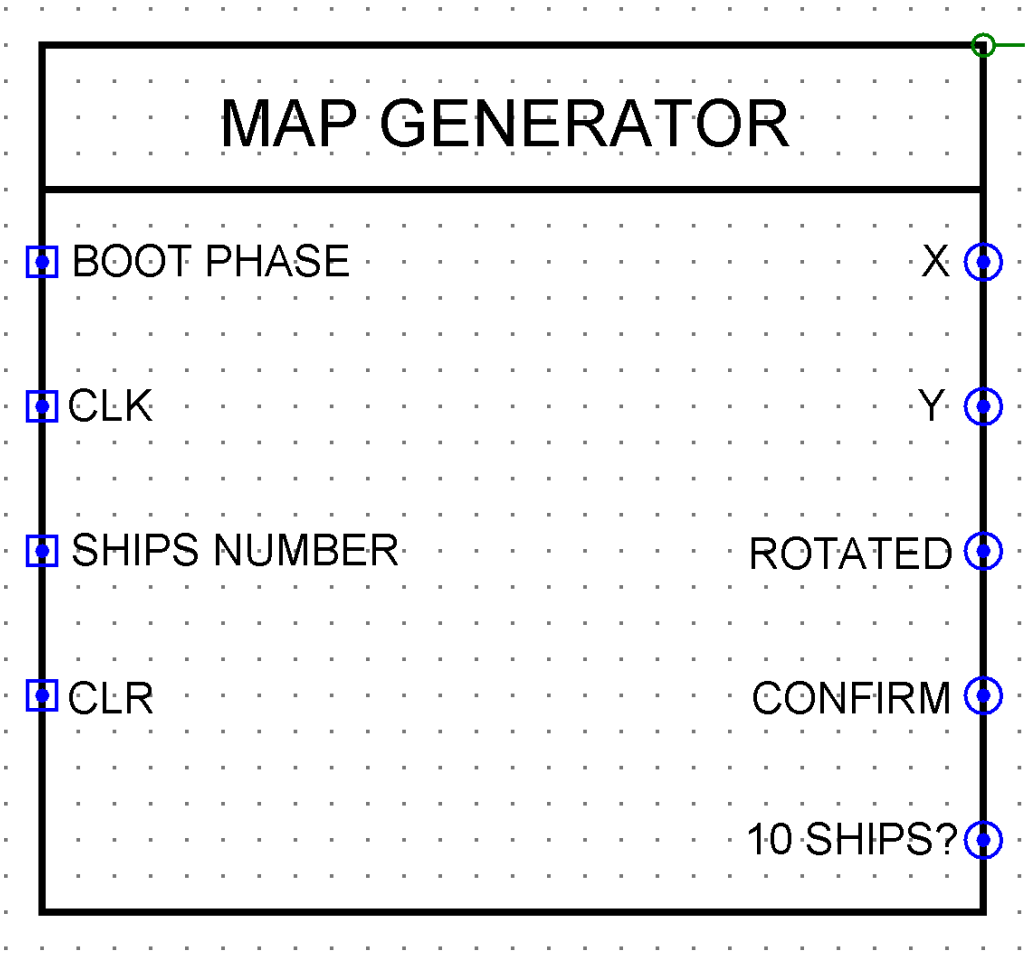


Rys. 51. Układ bots.circ (look under mask)

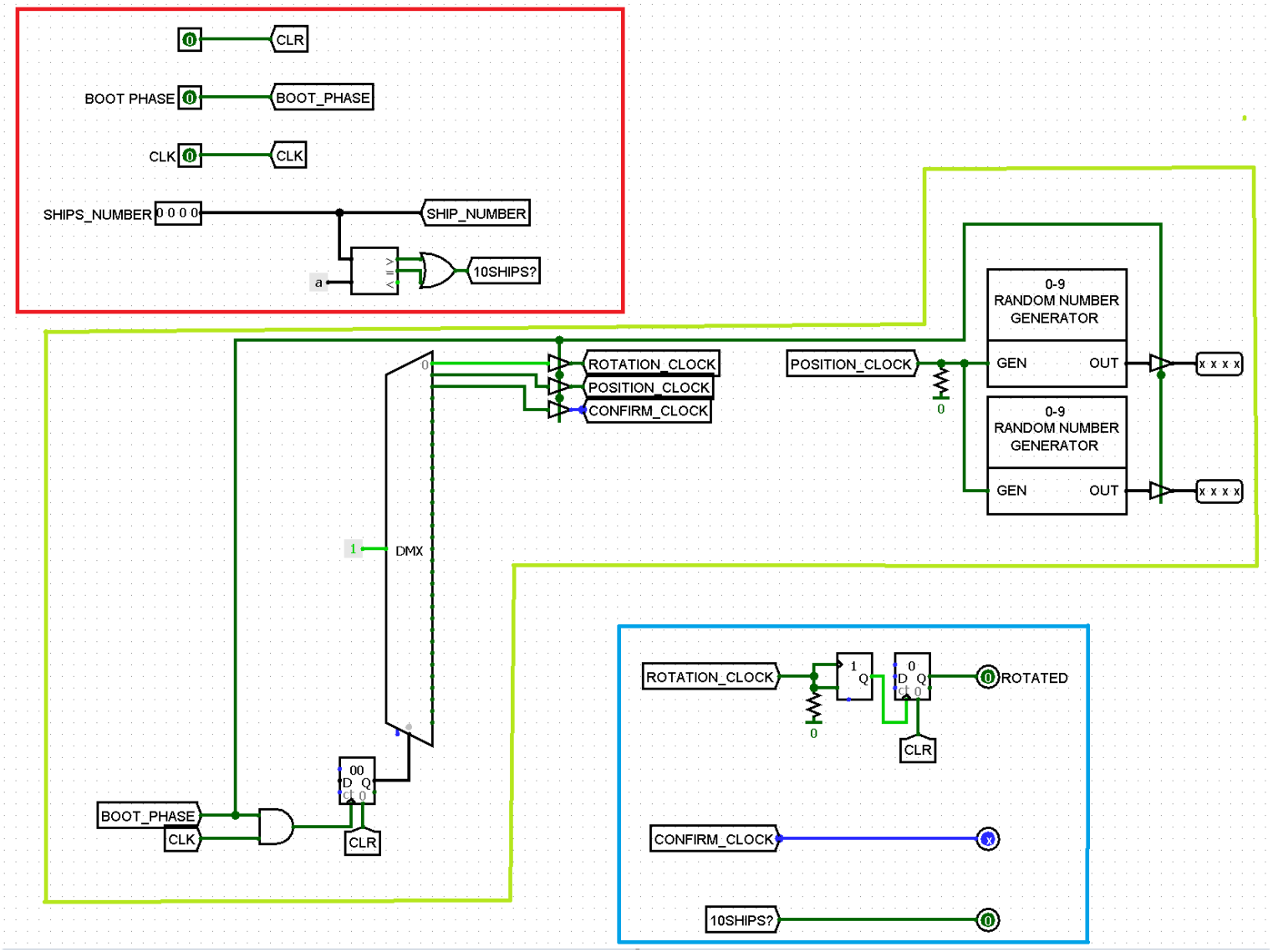
Cała logika układu z Rys. 51. opiera się na rozróżnieniu dwóch faz – BOOT_PHASE (faza uruchamiania, stawiania statków – generowania mapy) czy też SHOOT_PHASE (faza strzelania). Część układu zaznaczona kolorem niebieskim jest logiką decydującą, która faza trwa w danym momencie. Opiera się ona na dostarczeniu informacji o ilości statków aktualnie już umieszczonych na planszy. Flagi poniżej oraz po prawej obsługują jedynie poprawne wyświetlanie się wejść oraz wyjść bloku. W skład bloku wchodzą również: MAP_GENERATOR (map_generator.circ) oraz EASY BOT (bot_0.circ) opisane odpowiednio w punktach 4.12.1 oraz 4.12.2.

4.12.1. map_generator.circ

Układ zapewniający obsługę początkowej fazy bota – generowania mapy statków. Na wejście układ przedstawiony na Rys. 52. otrzymuje sygnał BOOT PHASE – określający czy aktywna jest faza tworzenia mapy, sygnał zegarowy CLK, SHIPS NUMBER określający aktualną ilość statków na planszy oraz CLR sygnał resetujący blok. Wyjściami układu są wyjścia analogiczne do całego bloku opisanego w punkcie 4.12. oraz jedno dodatkowe – czy została osiągnięta już maksymalna ilość statków (10).



Rys. 52. Układ map_generator.circ schemat bloku

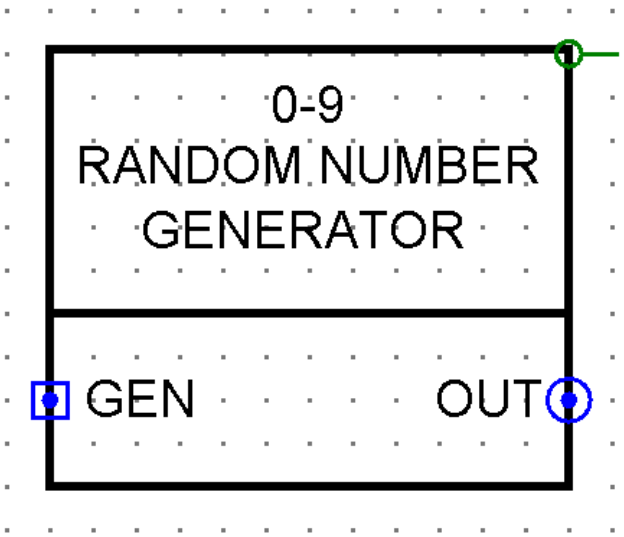


Rys. 53. map_generator.circ (look under mask)

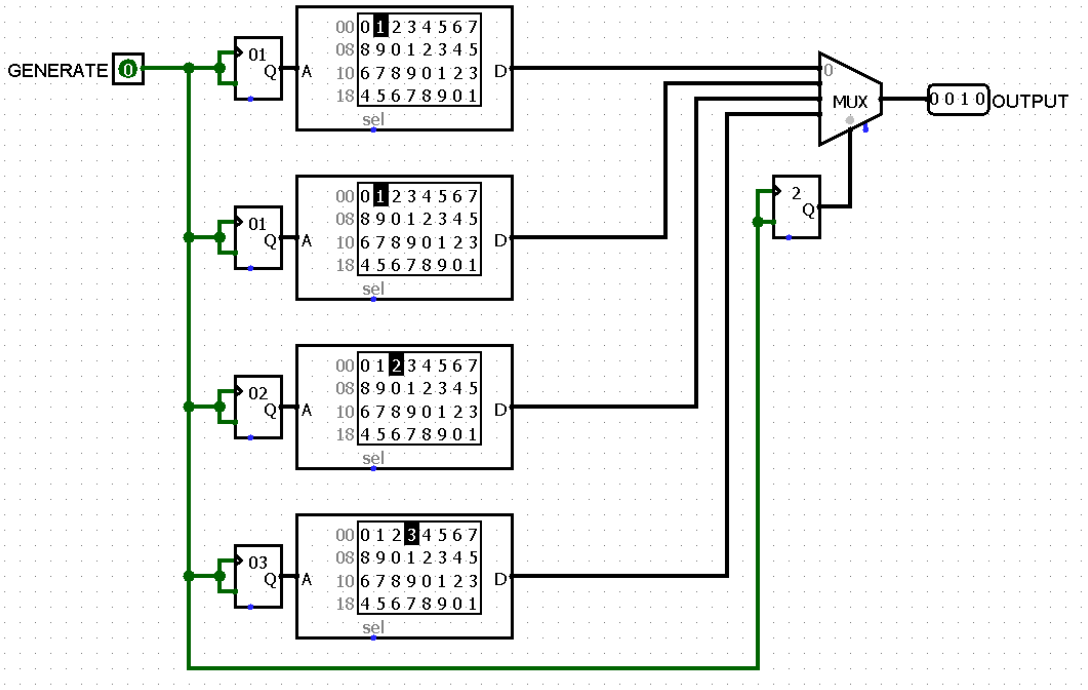
Część układu map_generator.circ zaznaczona kolorem czerwonym na Rys. 53. obsługuje jedynie wejścia bloku oraz występuje tu logika porównująca aktualną liczbę statków do dziesięciu. Kolorem niebieskim oznaczono obsługę wyjść oraz strukturę wybierającą pseudolosowo czy dany statek ma być obrócony czy też nie. Wyjściowy stan wstrzymywany jest przez dłuższy czas dzięki zastosowaniu licznika działającego jak flip-flop. Kolorem zielonym oznaczono najistotniejszą część tego układu – strukturę na wejściu otrzymującą sygnał BOOT_PHASE oraz CLK za pośrednictwem których obsługiwany jest licznik, który z kolei obsługuje demultiplekser. Na jego wyjściu są flagi dające komendy do kolejno wygenerowania pseudolosowego obrotu, pozycji oraz potwierdzenia strzału, nieużywane wyjścia służą zapewnieniu czasu oczekiwania (płynności z innymi blokami). Wylosowanie pseudolosowej pozycji odbywa się przy użyciu bloków random_number_generator_A.circ oraz random_number_generator_B.circ (oba opisane w punkcie 4.12.1.1.). Bloki te musiały zostać rozróżnione ze względu na różne seedy zapewniające różnorakie losowanie w tym samym ticku zegara.

4.12.1.1. random_number_generator_A.circ oraz random_number_generator_B.circ

Układ zapewnia generowanie liczby pseudolosowej z zakresu 0-9. Ze względu na to że podstawowy blok programu Logisim zapewniał jedynie ograniczenie co do ilości bitów w liczbach losowanych zastosowano przedstawione na Rys. 54. rozwiązanie. Wyjścia bloków podstawowych obsługują pamięć ROM, w której zapisane są oczekiwane cyfry. Jest to powtórzone czterokrotnie w celu otrzymania jak największej różnorodności losowania. Umieszczony na wyjściu multiplekser zapewnia pożądany rezultat.



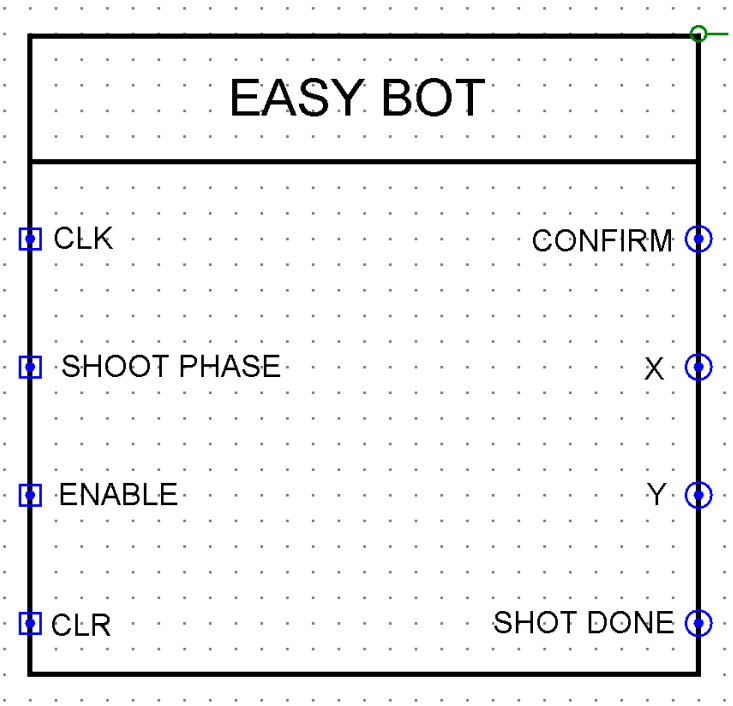
Rys. 54. Układ random_number_generator_A.circ schemat bloku



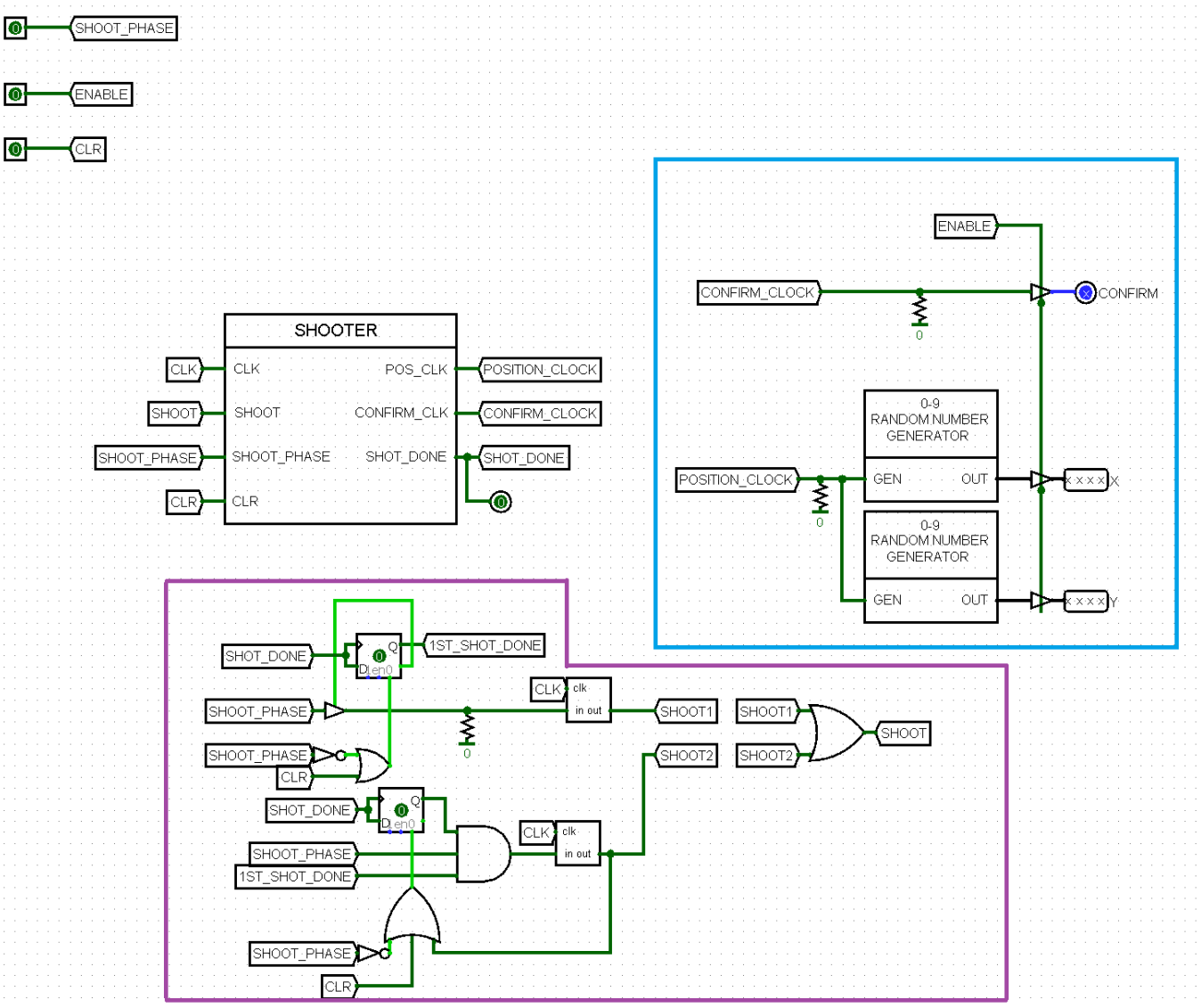
Rys. 55. Układ random_number_generator_A.circ (look under mask)

4.12.2. bot_0.circ

Układ zapewniający obsługę drugiej fazy bota – strzelania. Wejściami bloku przedstawionego na Rys. 56. jest sygnał zegarowy CLK, SHOOT PHASE określający fazę strzelania przez bota, ENABLE pozwalające na obsługę dwóch botów jednocześnie oraz CLR resetujące blok z zewnątrz. Wyjściami są: CONFIRM sygnał strzału bota w statek, X oraz Y to koordynaty odpowiednich osi (określające gdzie ma być strzał) oraz informacja zwrotna SHOT DONE określająca czy został już wykonany strzał.



Rys. 56. Układ bot_0.circ schemat bloku

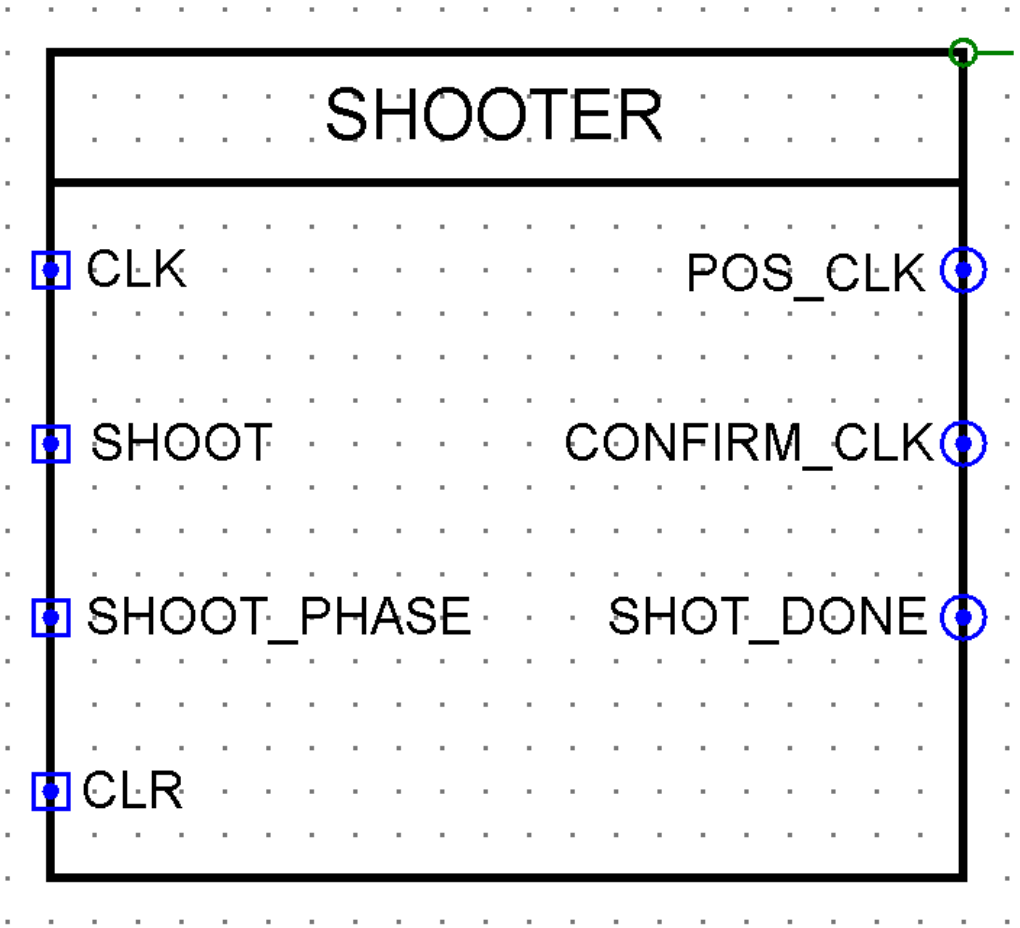


Rys. 57. Układ bot_0.circ (look under mask)

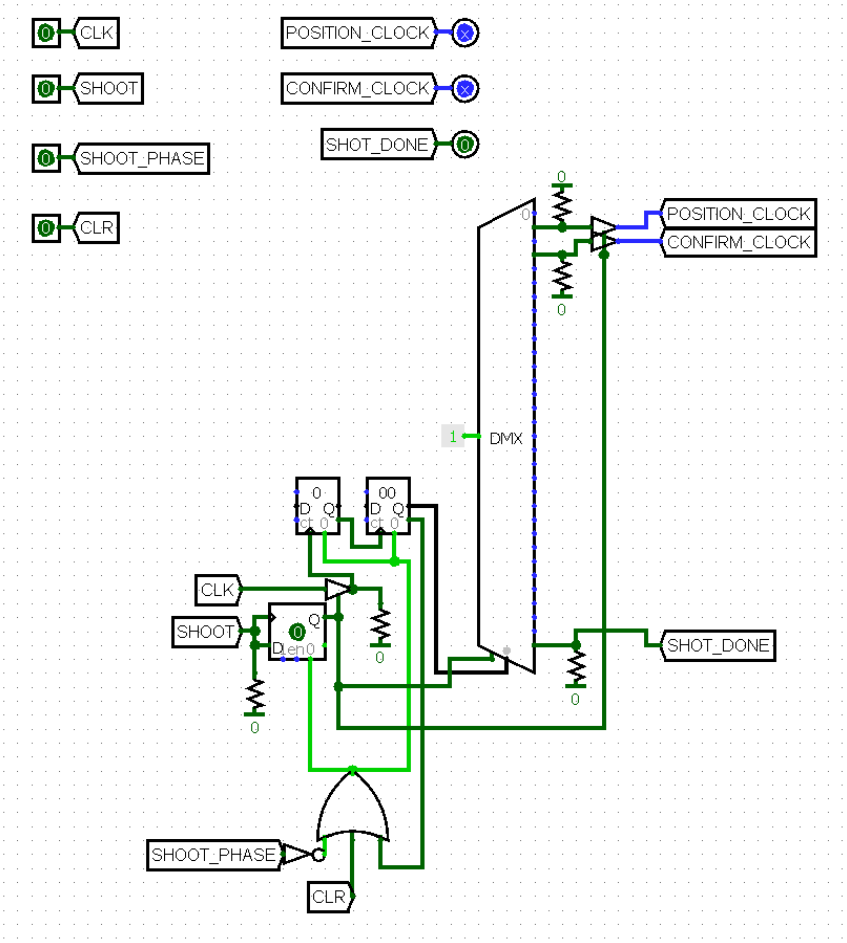
Na Rys. 57. kolorem niebieskim oznaczono część opisywanego układu odpowiedzialną za generowanie sygnału odnoszącego się do strzału oraz wybrania jego pozycji (analogicznie jak w punkcie 4.12.1.. Jedynie sygnał ENABLE obsługuje bufory blokujące sygnały wychodzące z bloku gdy jest taka konieczność. Kolorem fioletowym oznaczono logikę wykonującą pierwszy strzał (część górna) i każdy kolejny (część dolna). Ponadto blok EASY BOT wykorzystuje również układ SHOOTER (shooter.circ) opisany w punkcie 4.12.2.1..

4.12.2.1. shooter.circ

Układ najważniejszy, obsługujący samą mechanikę strzału. Na wejście bloku przedstawionego na Rys. 58. wchodzi: sygnał zegarowy CLK, komenda SHOOT (określająca czy ma być wykonany strzał), SHOOT PHASE opisane w punkcie nadrzędnym 4.12.2. oraz reset zewnętrzny CLR. Wyjściami układu jest sygnał przedstawiający decyzję o wykonaniu, bądź też nie, POSITION_CLOCKa, analogicznie CONFIRM_CLOCK – dezycja o strzale oraz SHOT_DONE sygnał określający to, że strzał został już wykonany.



Rys. 58. Układ shooter.circ schemat bloku

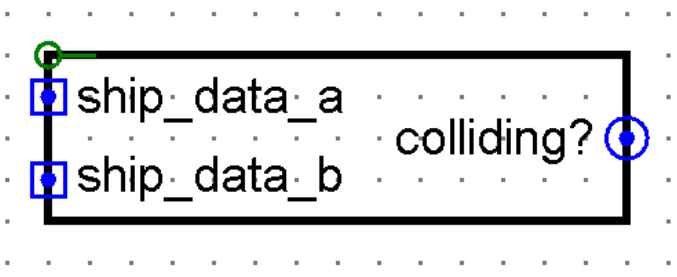


Rys. 59. Układ shooter.circ (look under mask)

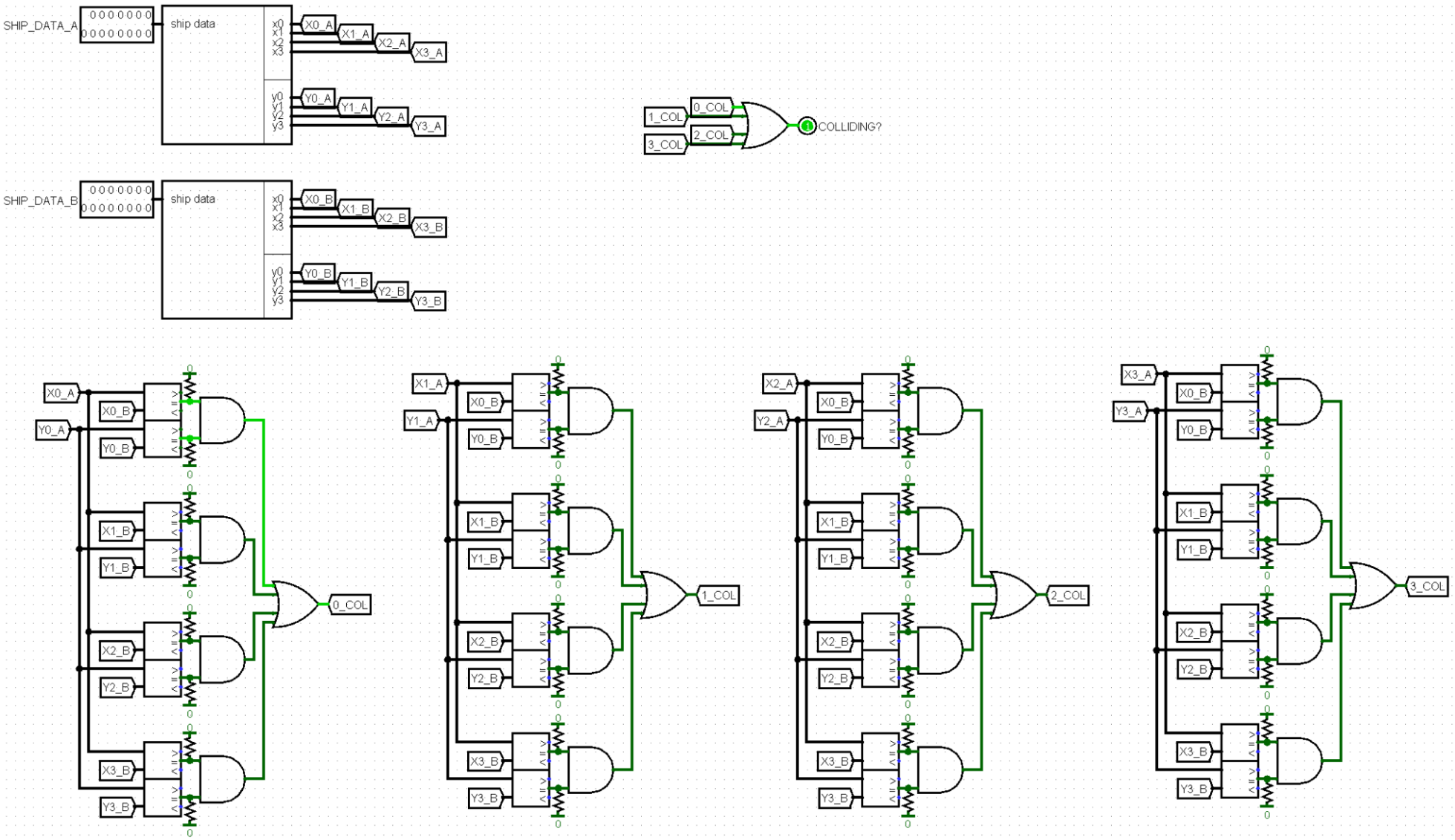
Logika układu przedstawionego na Rys. 59. opiera się na przechodzącym stanie wysokim sygnału SHOOT na podtrzymujący go przerzutnik typu D, dzięki czemu na jego wyjściu w stanie wysokim otwiera się bufor. Implikuje to przejście sygnału zegarowego do liczników (ich głównym zastosowaniem jest obsługa demultipleksera, wykorzystanie dwóch liczników służy jedynie dzieleniu sygnału zegarowego – rozrzedzeniu go). Dzięki tej budowie proces strzału wykonuje się wolniej. Przepełnienie pamięci ("koniec" demultipleksera) implikuje wyzerowanie przerzutnika i nadanie sygnału, że strzał został wykonany.

4.13. ships_colliding.circ

Układ obsługujący wykluczenie nachodzenia się na siebie statków, w celu nie nachodzenia się ich na siebie w trakcie układania ich na planszy. Wejściami układu przedstawionego na Rys. 60. są jedynie pamięci określające dwa różne statki, na wyjściu natomiast znajduje się stan – wysoki gdy statki kolidują lub niski gdy nie.



Rys. 60. Układ ships_colliding.circ schemat bloku

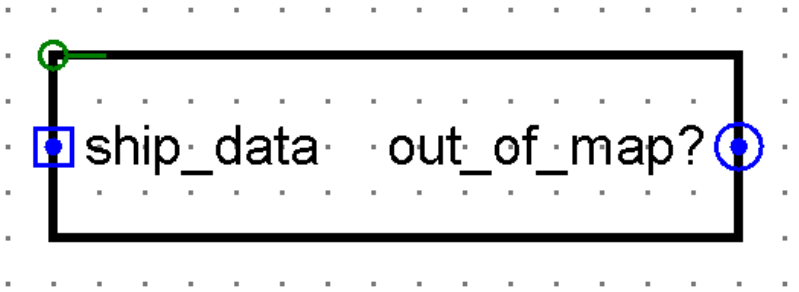


Rys. 61. Układ ships_colliding.circ (look under mask)

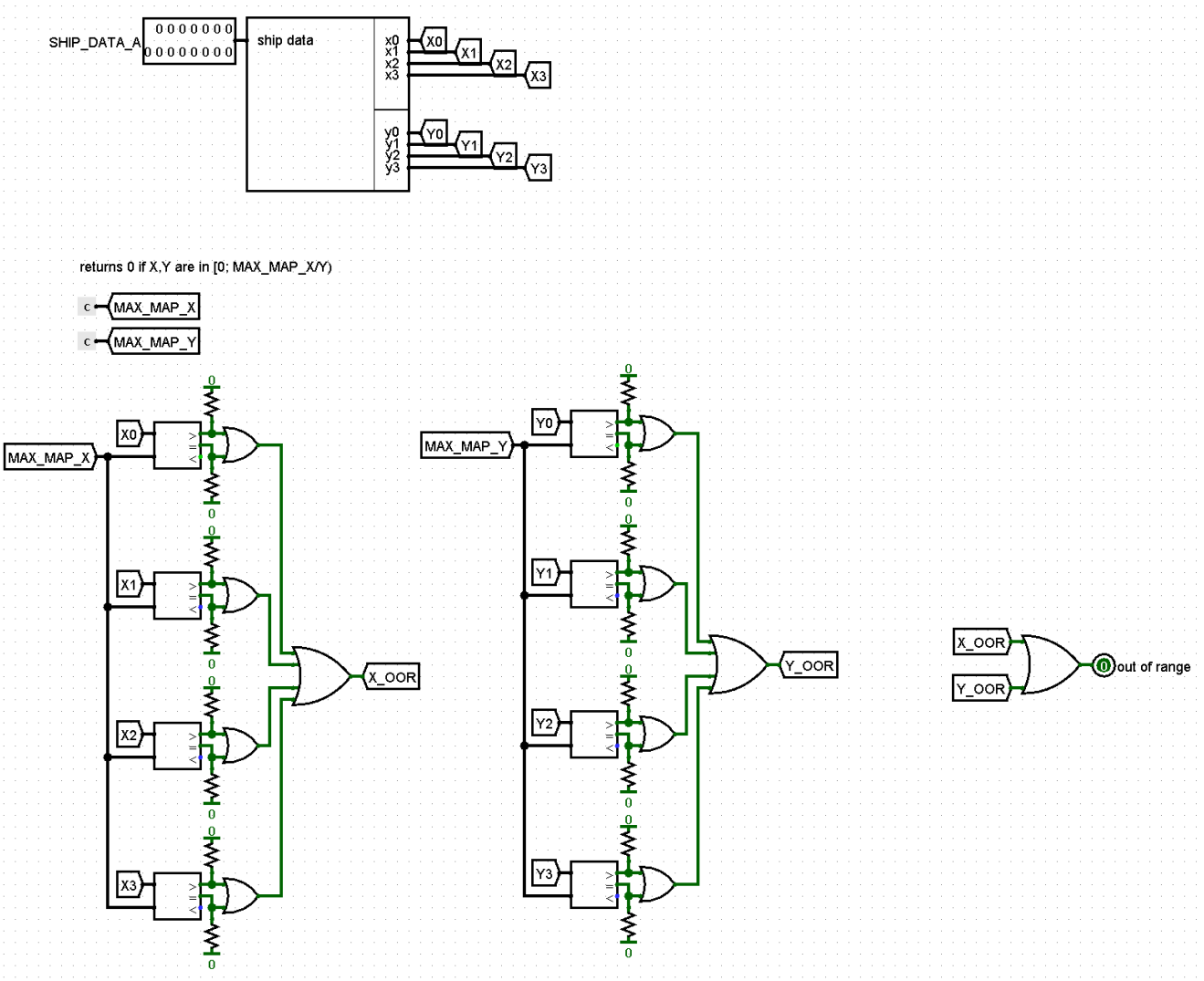
Logika układu ships_colliding. circ opiera się na użyciu struktur logicznych utworzonych z bramek AND, OR oraz z komparatorów w celu porównania rozbitych danych o statkach wychodzących z bloków ship data.

4.14. ships_out_of_map.circ

Układ obsługujący teoretyczne wyjście poza mapę statku przy jego usytuowaniu na planszy. Układ przedstawiony na Rys. 62. na wejściu otrzymuje jedynie pamięć konkretnego statku a na wyjściu określa czy jest on poza przeznaczoną do gry mapą (stan wysoki), czy też nie (stan niski).



Rys. 62. Układ ships_out_of_map.circ schemat bloku

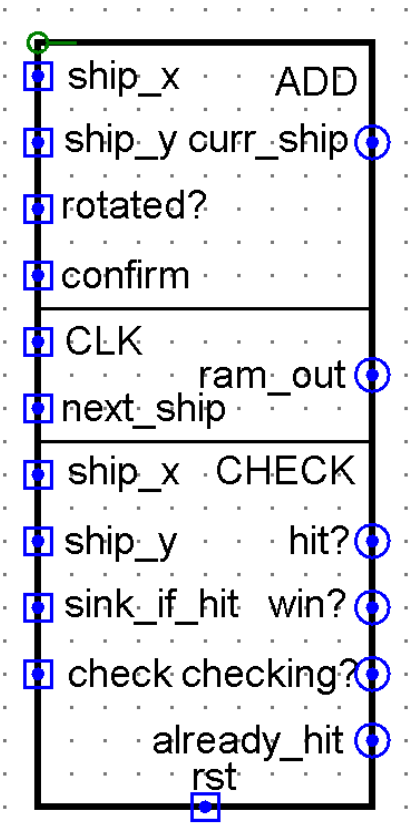


Rys. 63. Układ ships_out_of_map.circ (look under mask)

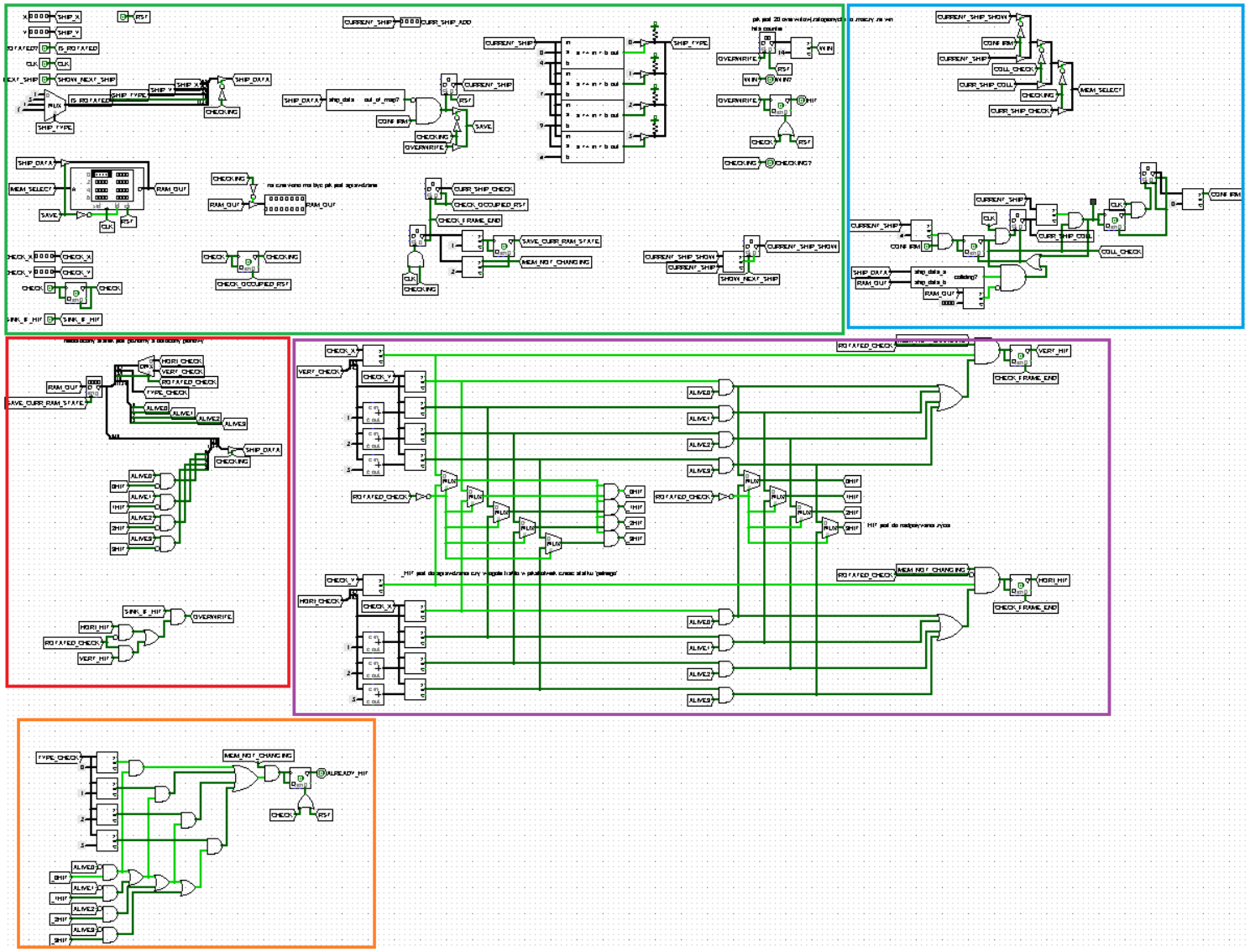
Logika układu ships_out_of_map.circ, analogicznie jak ships_colliding. circ opiera się na użyciu struktur logicznych utworzonych z bramek AND, OR oraz z komparatorów w celu porównania rozbitych danych o statkach wychodzących z bloków ship data, w celu identyfikacji ewentualnego wyjścia poza mapę.

4.15. memory_managment.circ

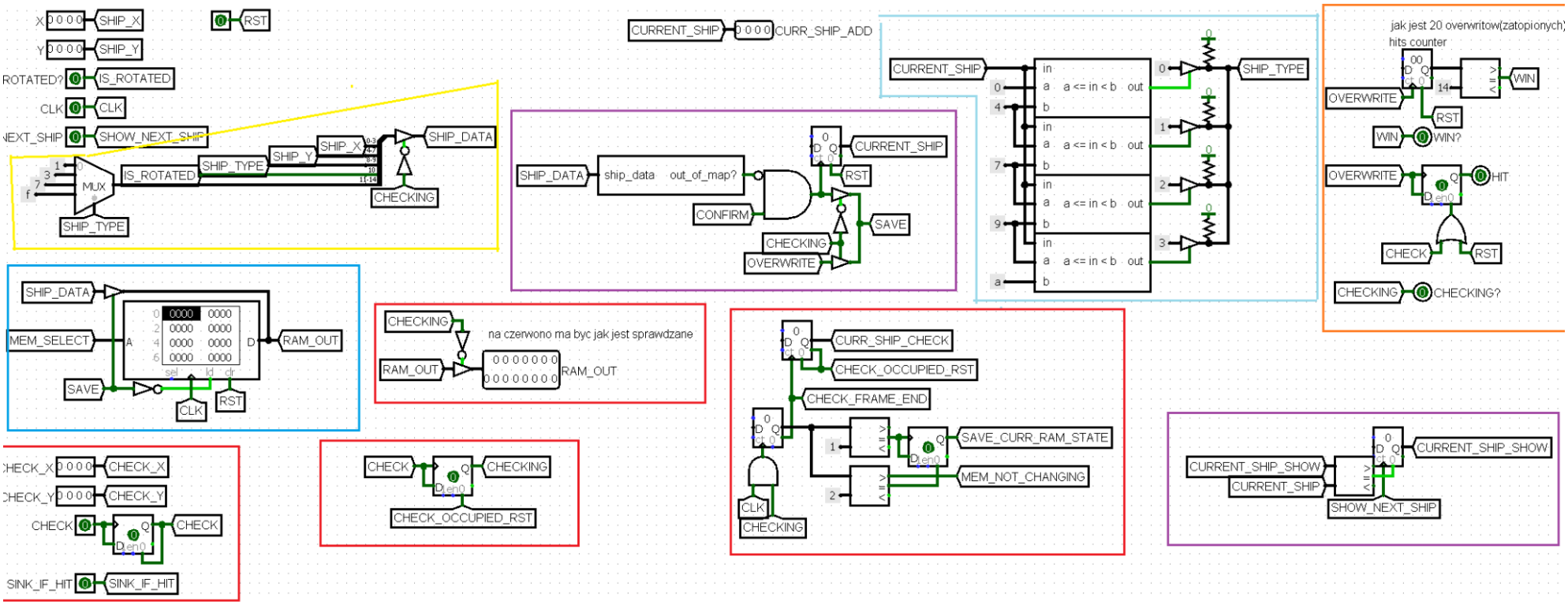
Układ obsługujący całe zarządzanie statkami – dodawaniem ich, sprawdzaniem wszystkich kolizji (nachodzenia się na siebie oraz ewentualnych wyjść za mapę), nadpisywaniem życia statkom w momencie gdy następuje trafienie. Cały blok przedstawiony na Rys. 64. podzielony jest właśnie na te 3 zadania. Wejściami bloku dodającego są: ship_x oraz ship_y określające koordynaty miejsca zapisu na mapie, rotated? czy statek ma być obrócony (statek nieobrócony jest poziomy), confirm deklaracja zapisu. Natomiast jego wyjściem jest curr_ship potrzebny do określenia aktualnego statusu gry (tutaj dodawanie). Część druga na wejściu otrzymuje sygnał zegarowy CLK oraz sygnał next_ship odpowiedzialny za przekazanie informacji o obsługę kolejnego statku przez pamięć RAM – jedynie wyświetlanie. Część sprawdzająca na wejściu otrzymuje koordynaty ship_x oraz ship_y miejsca do sprawdzenia, sink_if_hit odpowiedzialne za zatapianie jeśli pole zostało trafione, check wydające komendę sprawdzenia w danym momencie. Wyjścia tej części: hit? – czy statek został trafiony, win? – czy nastąpił koniec gry, checking? – następuje sprawdzanie oraz already_hit – sygnał informujący o tym, że dany sektor został już trafiony wcześniej.



Rys. 64. Układ memory_managment.circ schemat bloku

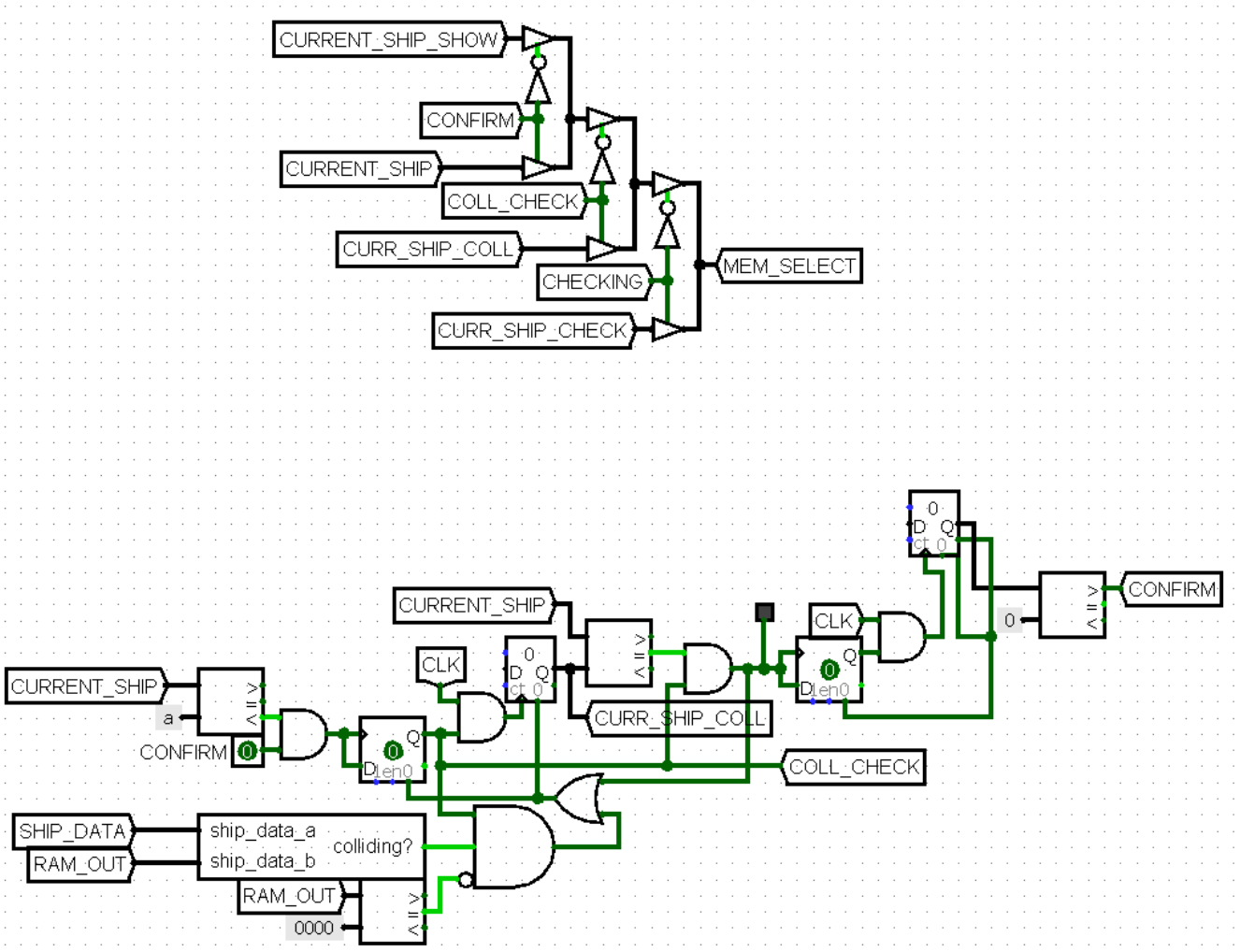


Rys. 65. Poglądowy układ memory_managment.circ, w celu zobrazowania położenia poszczególnych części bloku (look under mask)



Rys. 66. Część układu (przybliżenie) memory_managment.circ przedstawionego na Rys. 65. zaznaczonego kolorem zielonym

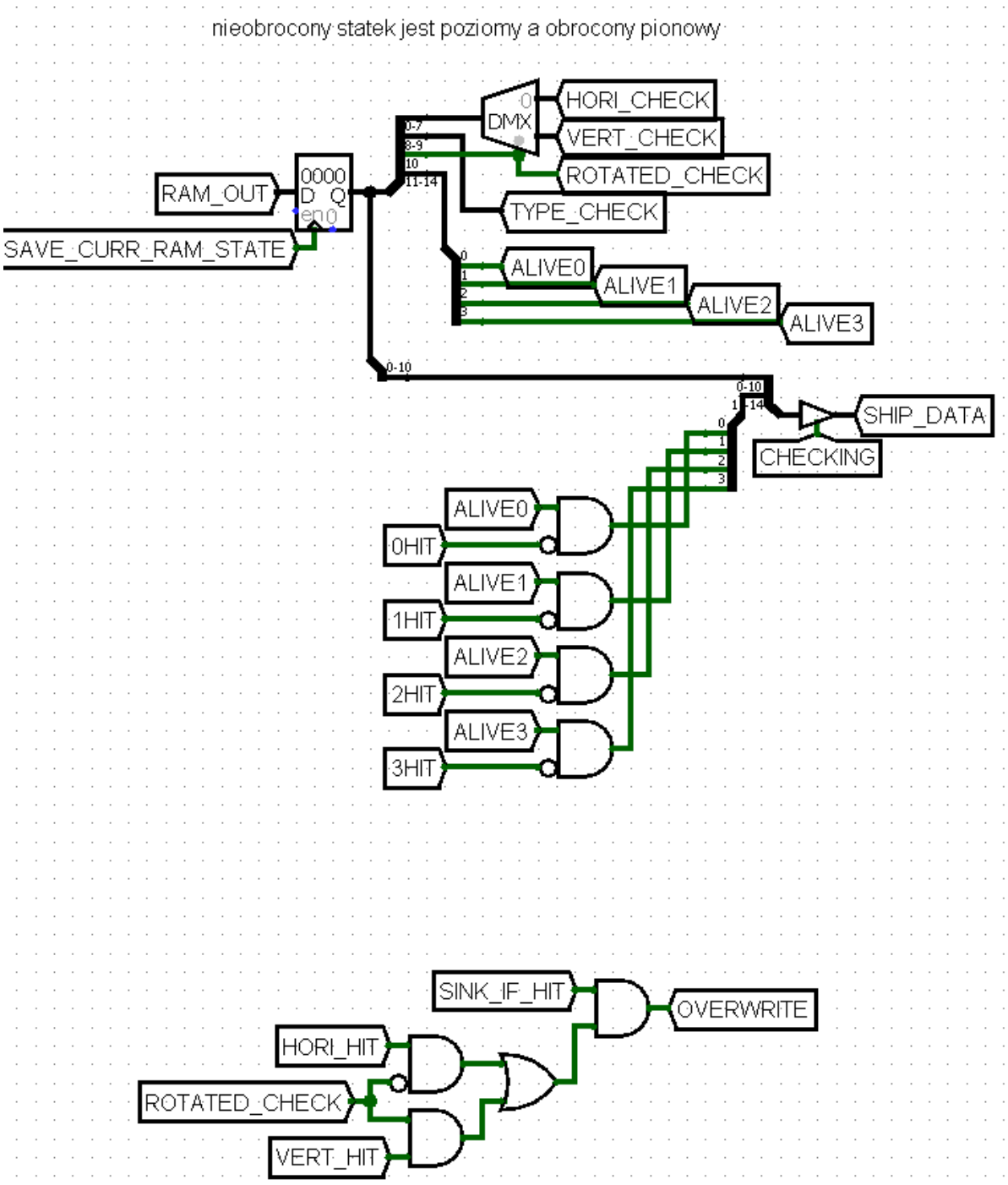
Część układu zaznaczona kolorem niebieskim na Rys. 66. jest sercem całego bloku – pamięcią RAM. Nie można jednocześnie zapisywać oraz odczytywać z tej pamięci, w tym celu zastosowano flagi. MEM_SELECT odpowiedzialna jest za wybór komórki, na której odbywa się aktualna praca, tworzona jest przy pomocy struktury logicznej znajdującej się na górze Rys. 67., w jej skład wchodzą: CURR_SHIP_CHECK zapewnia obsługę wszystkich 10 dostępnych statków, CURR_SHIP_COLL obsługuje kolizje, CURRENT_SHIP obsługuje dodawanie statków – który statek dodajemy, w którym miejscu (tworzona jest ona w układzie zaznaczonym kolorem zielonym, zawiera bramkę AND, licznik oraz ship_out_of_map.circ), a CURRENT_SHIP_SHOW obsługuje wyświetlanie (tworzona jest w części zaznaczonej kolorem fioletowym, licznik liczy do 16, resetuje się gdy wyjście Q licznika osiągnie wartość równą wartości flagi CURRENT_SHIP – aktualnej ilości statków zapisanej w pamięci). Kolorem żółtym oznaczono strukturę odpowiedzialną za zebranie informacji o aktualnie dodawanym statku. SHIP_X, SHIP_Y, IS_ROTATED są bezpośrednio pobierane z wejść, a pozostałą informację uzyskujemy na podstawie wyjścia układu zaznaczonego kolorem jasnoniebieskim - SHIP_TYPE, który otrzymywany jest z prostych warunków logicznych, porównujący numer aktualnie dodawanego statku do stałych wartości. Część oznaczona kolorem pomarańczowym obsługuje status WIN? określający zwycięstwo po wykonaniu 20 nadpisań oraz wyjście trafienia.



Rys. 67. Część układu (przybliżenie) memory_managment.circ przedstawionego na Rys. 65. zaznaczonego kolorem niebieskim

Dodając nowy statek obsługujemy dolny układ z Rys. 67.. Początkowo sprawdzane jest czy aktualna ilość statków jest mniejsza od wartości oczekiwanej 10, jeżeli to jest spełnione załączany jest stan wysoki na wyjściu przerzutnika, który, po przejściu przez bramkę AND, umożliwia podanie sygnału CLK do licznika odpowiedzialnego za wybieranie pamięci do obsługi kolizji – zanim cokolwiek zacznie się dodawać sprawdzane czy jest w pamięci jest już jakiś statek, który kolidował by z aktualnie dodawanym. Jeśli okaże się, że zapisywany aktualnie statek koliduje z wcześniej już powstałym, cały proces zostaje przerwany i nic się nie dzieje, jeżeli natomiast tak nie jest następuje zapis – załączany jest kolejny licznik, który, z racji tego, że RAM potrzebuje zbocze narastające, zatrzymuje się w tym miejscu na dwa ticki zegara. Część oznaczona kolorem fioletowym na Rys. 66. odpowiedzialna jest za sam zapis. Początkowo sprawdzane jest czy teoretyczny statek do zapisu nie wyjdzie za mapę gry przy pomocy układu ship_out_of_map.circ, następnie na fladze zapisu SAVE pojawia się stan wysoki.

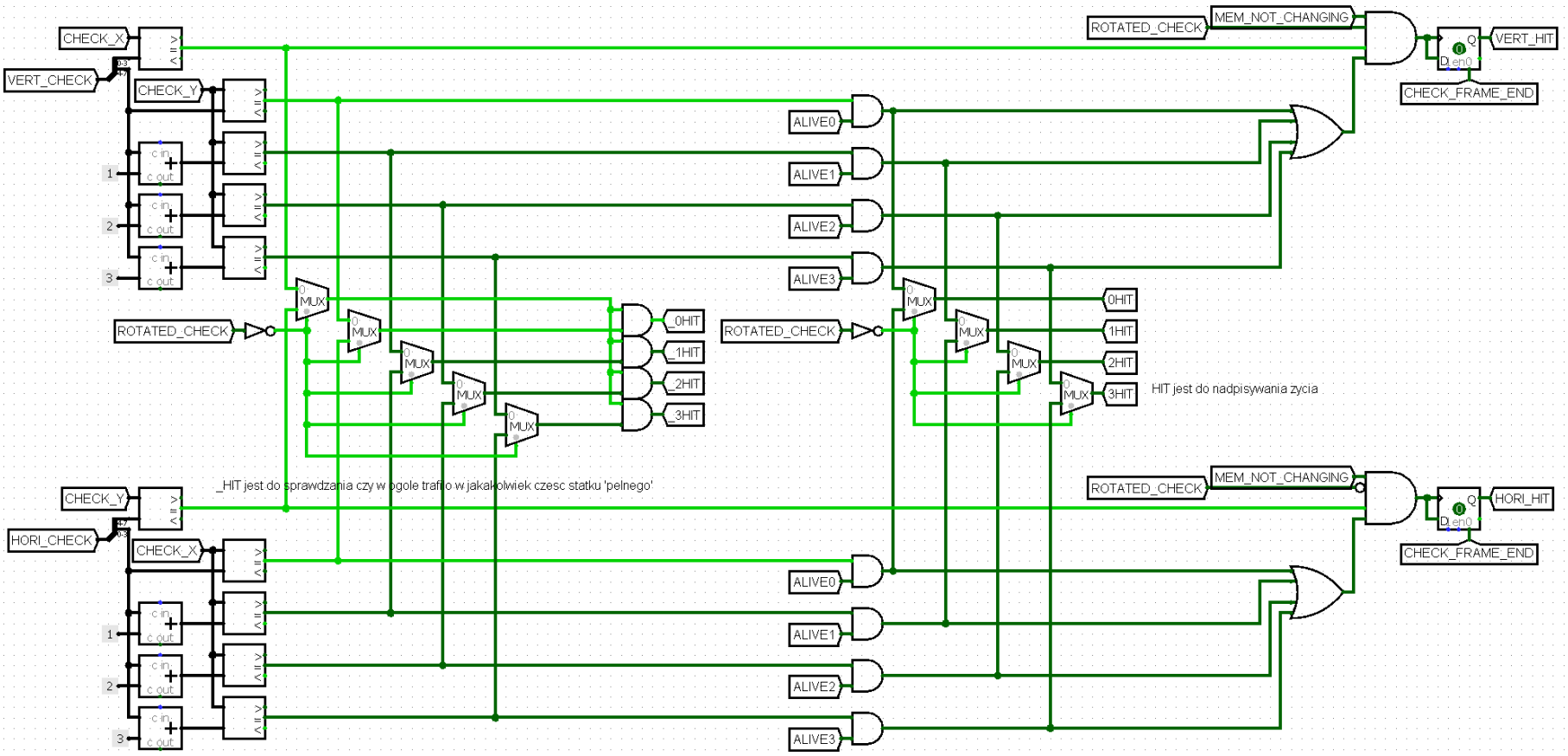
Na Rys. 66. kolorem czerwonym zaznaczono moduły odpowiedzialne za fazę sprawdzania tj. od lewej: wyprowadzenie wejść potrzebnych do sprawdzenie kolizji, obsługa utrzymywania sygnału sprawdzania CHECKING przy użyciu przerzutnika typu D, dalej jeśli jest aktywny CHECKING to rozpoczęte jest naliczanie na liczniku (sprawdzenie pojedynczego statku zajmuje 7 cykli zegara), aktualnie sprawdzany statek przechowywany jest w rejestrze. Góry licznik sprawdza, który statek aktualnie jest sprawdzany flaga CURR_SHIP_CHECK przekazywana jest do modułu którego wyjściem jest MEM_SELECT (górna część Rys. 67.).



Rys. 68. Część układu (przybliżenie) memory_managment.circ przedstawionego na Rys. 65. zaznaczonego kolorem czerwonym

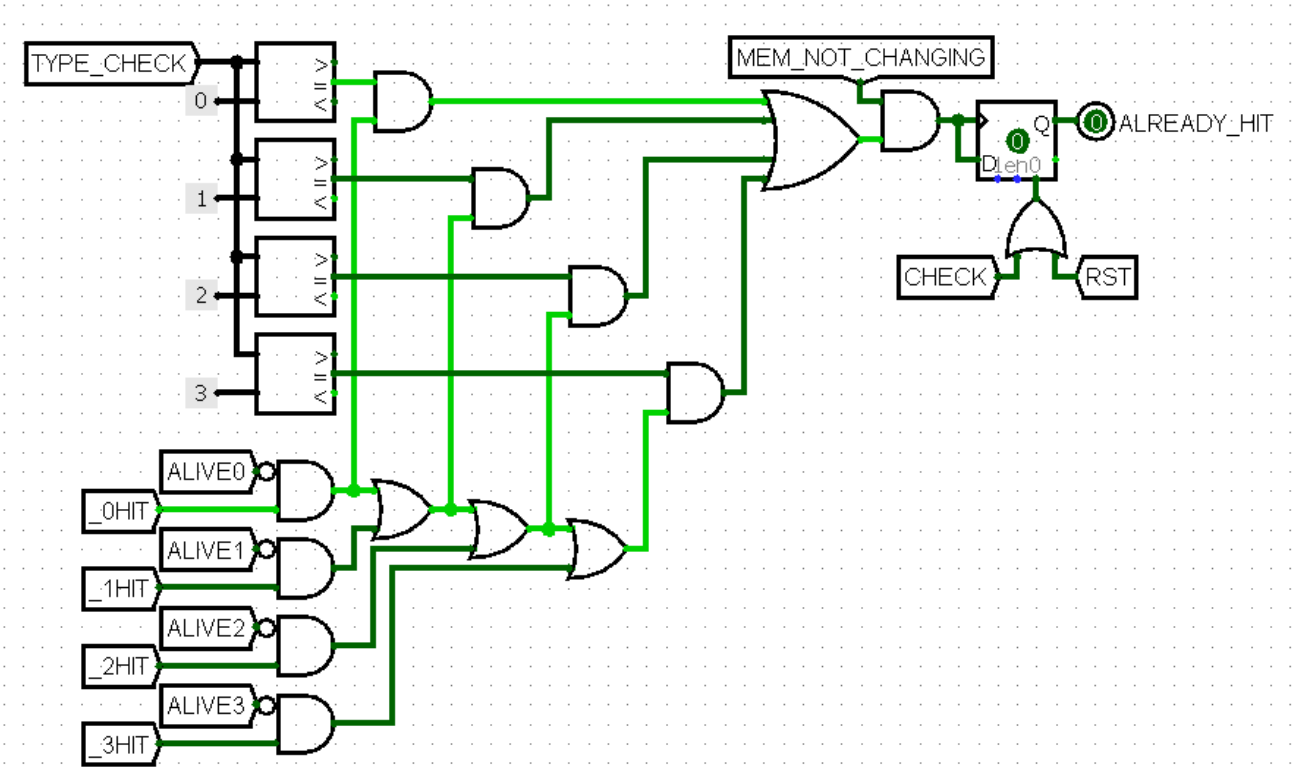
Górna część modułu z Rys. 68. rozdziela wcześniej przygotowaną pamięć na części gotowe do sprawdzenia w układzie na Rys. 69. tj.: HORI_CHECK, VERT_CHECK, ROTATED_CHECK, TYPE_CHECK oraz flagi ALIVE. Gałąź poniżej służy nadpisywaniu po trafieniu, wedle zależności jeśli część statku była żywa i została trafiona to jest nieżywa.

Dolna część służy do nadpisywania pamięci – jeśli włączona jest flaga SINK_IF_HIT oraz jest jeśli nastąpiło uderzenie horyzontalne, a opisywany statek jest horyzontalny to należy nadpisać. Sytuacja wygląda analogicznie przy wertykalnym.



Rys. 69. Część układu (przybliżenie) memory_managment.circ przedstawionego na Rys. 65. zaznaczonego kolorem fioletowym

Na sprawdzeniu pamięci kolizji wertykalnie sprawdzanie są jedynie koordynaty OY, gdyż OX dla tego typu statków jest stałe. Początkowo sprawdzane jest czy koordynata x jest równa zadanej, następnie koordynaty OY rozkładane są na części pamięci odpowiedzialne za poszczególne piksele statku, sprawdzane jest czy pierwszy y jest równe zadanemu oraz czy kolejne po odpowiednim przesunięciu. Jeśli x, y się zgadzają a sprawdzany segment jest żywy to znaczy, że trafiono w statek. Następnie trafienia pionowe wychodzą na przerzutnik typu D. Część dolna sprawdzenia horyzontalnego jest analogiczna. Wyjścia ustawione są jako flaga – jeśli nastąpiło już trafienie statku danego to jest to zapisywane do końca. Cztery wyjścia _xHIT używane są dalej w części układu przedstawionej na Rys. 70.. Mówią one jedynie o tym czy koordynaty sprawdzenia są równe któremuś z koordynatów segmentu statku.



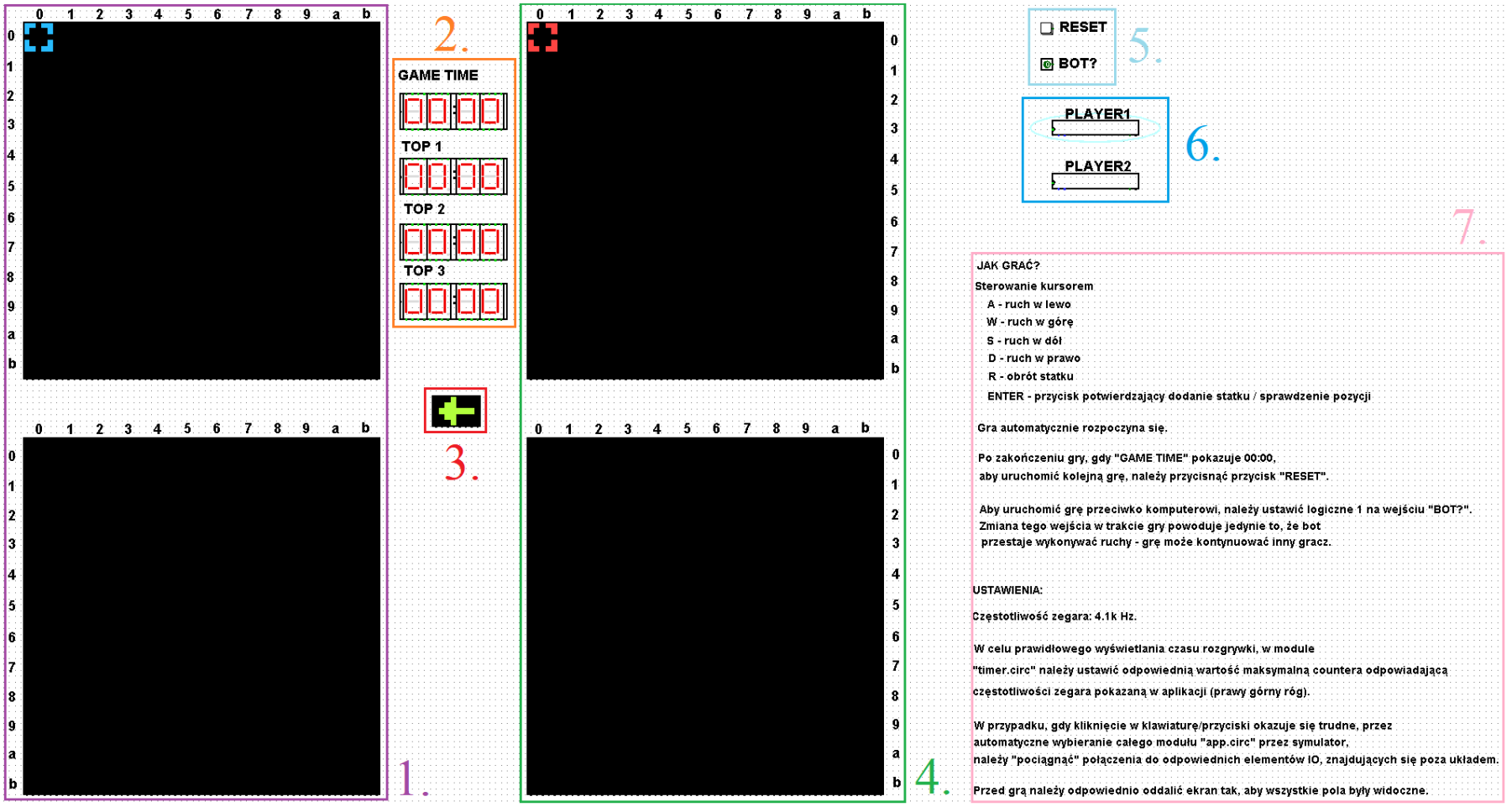
Rys. 70. Część układu (przybliżenie) memory_managment.circ przedstawionego na Rys. 65. zaznaczonego kolorem pomarańczowym

Jeśli koordynaty x oraz y z bloku z Rys. 69. się zgadzają oraz omawiany segment statku nie jest żywy (jeżeli został już trafiony) to na wyjściu otrzymuje 1, następnie każdy segment (oraz ich różnorakie połączenia) sprawdzane, porównywane są do typu statku jakim są i na tej podstawie na wyjściu układu na Rys. 70. otrzymuje się flagę ALREADY_HIT.

5. OMÓWIENIE INTERFEJSU UKŁADU

1) Ekrany gracza 1, górny pokazuje statki gracza 1 i trafienia/nietrafienia gracza 2, dolny pokazuje strzały gracza 1 w statki gracza 2.
2) Zegar aktualnego czasu gry i wyświetlanie najlepszych (najszybszych) wyników.
3) Wskaźnik określający aktualną kolej wskazanego gracza.
4) Ekrany gracza 2, górny pokazuje statki gracza 2 i trafienia/nietrafienia gracza 1, dolny pokazuje strzały gracza 2 w statki gracza 1.
5) Przyciski pozwalające na reset gry oraz włączenie/wyłączenie bota.
6) Wejścia do sterowania dla obu graczy.
7) Instrukcja obsługi oraz ustawienia rekomendowane.

Rysunek znajduje się na kolejnej stronie.

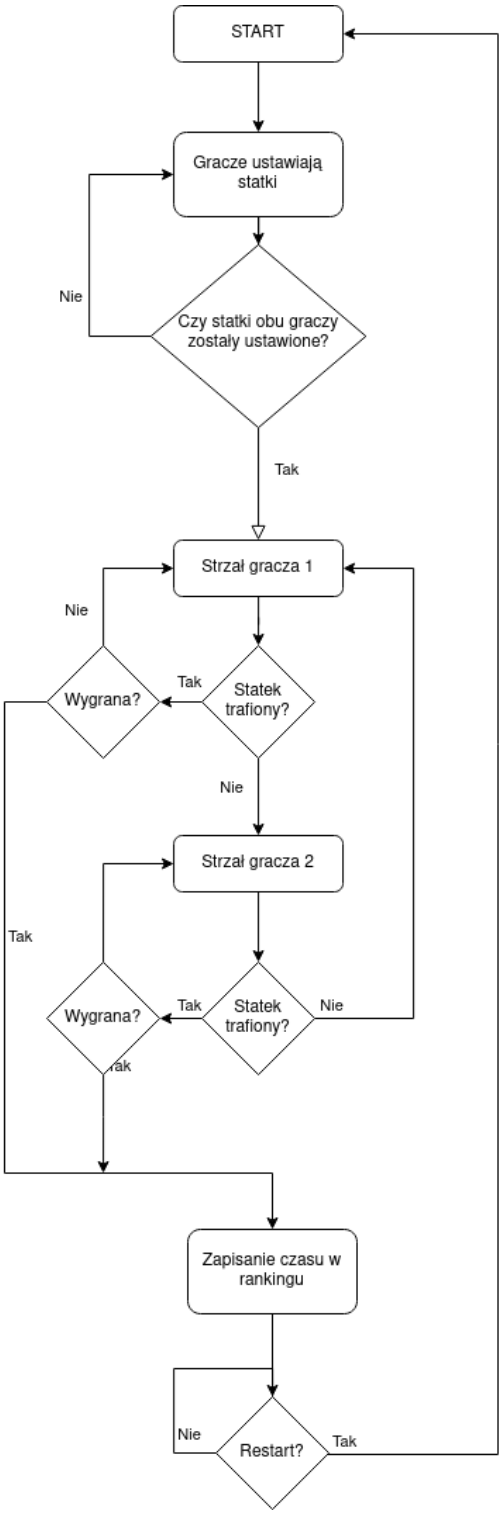


Rys. 71. Omówienie interfejsu układ

6. MOŻLIWOŚCI DALSZEGO ROZWOJU/NIEZREALIZOWANA FUNKCJONALNOŚĆ

Wprowadzenie obsługi bota o różnych stopniach trudności. Ekran początkowy i końcowy.

7. SCHEMAT LOGICZNY UKŁADU



W przypadku załączenia trybu bot, graczem drugim steruje bot. Możliwa jest zmiana trybów w dowolnym momencie rozgrywki.

8. NAGRANIE WIDEO

https://youtu.be/4EY9RFd9W2g