Symbol Generator Blocks

# Abbreviations

MP= Message Pack (includes [SOF, Type, Address, Data Length ,Payload ,CRC ,EOF].

MPD= Message Pack Data (1 pack of 8 bits that is transmitted in the Payload field. There can be more than 1 MPD in Payload field and the number of MPD is represented by the field Data Length.

OPU= the block Opcode\_Unite.

RM= the block Read\_Manager.

Remarks:

1. we figured that we need only 23 bits to create 1 opcode (equal to 3 MPD).

Calculation:

* command =
* Com\_type – add or remove (1 bit)
* x coordinate (4 bits)
* y coordinate (5 bits)
* com\_add : address of the symbol in SDRAM (11 bits because: SDRAM depth is 2^12, SDRAM width is 2^8, and there are 4 banks represented by 2 more bits. However, we need only the depth of the SDRAM to represent the address of the symbol, because we use all the width of the SDRAM (we use the whole row) therefore 12 bits for address.

However, the symbol is saved in 2 rows in the SDRAM therefore the LSB is known – (even or odd row) , so we actually need 11 bits.

* 2 bits to represent the Bank we use in the SDRAM (total of 4 Banks)

**TOTAL = 23 bits for the command**

1. We want to save 1 symbol that will be the color black to represent the removing of a symbol. This unique symbol will be saved at the first row of the SDRAM (at address "000000000000,00000000,00" = (row,col,bank).

# Conceptual Description – Data Flow

The data flow in the Symbol Generator block is as follows:

1. The command packages (opcodes) from the SW are transferred through the Wishbone protocol and received in the Opcode Unite block. The Opcode Unite block unites the commands, which come in byte packets, and transfers them to the Command FIFO.
2. The Command FIFO receives the united opcodes and stores them until a new video frame is initialized by the VESA Generator block. The reason we want to store the opcodes until the new frame is because we don't want to override the current display which is stored in the RAM. Therefore, we store the opcodes until a new frame is initialized, and then we can transfer the opcodes to the RAM.

When opcodes from Command FIFO are transferred to the RAM, the Command FIFO is flushed, in order to be empty and ready for the next data packets to arrive.

1. The RAM represents the current display. Each row in the RAM indicates a 1 (x,y) block on the display.

The RAM receives the opcodes from the Command FIFO, and stores each opcodes in the relevant row in the RAM (the relevant row is calculated using fields in the Opcode). The RAM stores the addresses in the SDRAM, in which the wanted symbols are saved.

When all the opcodes were updated in the RAM, a signal for the Read Manager block is asserted.

1. When the VESA Generator requests video data, the Read Manager initializes read transaction from the SDRAM through the Wishbone master interface. The addresses of the symbols in the SDRAM are stored in the RAM. The Read Manager manages the read from the SDRAM using calculations of the row and the column of the address in the SDRAM. The data which is received from the SDRAM in the read transaction is stored in one of the FIFOs in the Read Manager block: FIFO-A or FIFO-B.

Here we use our algorithm to calculate the address in the SDRAM (row and column) and how to toggle between the two FIFOs (A and B).

1. Using a MUX, the relevant data from one of the FIFOs is transferred to the Dual Clk FIFO, in order to pass between the two clock domains:
2. MHz of the FPGA 🡪 40Hz of the VESA on the display
3. From the Dual Clk FIFO, video data is delivered to the VESA Generator block. And then to the display.

To add a block diagram of the conceptual description of the Symbol Generator block!

# Detailed Description

## Opcode Unite - OPU

**Goal:** OPU unites every 3 packs MPD into 1 opcode by a Finite State Machine (FSM) and writes data to Opcode FIFO.

**Functionality:** When Wbs\_adr\_i [9..0] = OPU\_address, then OPU is being activated.

**Interface Signals:**

Detailed explanation:

Input:

1. Clk\_133 : the main clock to which all the internal logic of the Symbol Generator block is synchronized.
2. Reset : asynchronous reset of the Symbol Generator block.
3. Wbs\_adr\_i [9..0] : this signal is sent from the WBS. It indicates the address of the Display Controller Top. With it we can know if our blocks inside Display Controller are requested – TBD with Beeri
4. Wbs\_tga\_i [9..0] : this signal is sent from the WBS. It indicates how many words are there that are transmitted to us (in other words, it indicate the field Data\_Length in the message pack.
5. Wbs\_dat\_i [7..0] : this signal is sent from the WBS. It indicates the data itself from the field Payload in the message pack.
6. Wbs\_cyc\_i : this signal is sent from the WBS. It indicates the signal cycle required by the Wishbone protocol.
7. Wbs\_stb\_i : this signal is sent from the WBS. It indicates the signal strobe required by the Wishbone protocol.

Output:

1. Wbs\_ack\_o

The acknowledge signal required by the Wishbone protocol.

1. Wbs\_stall\_o

The stall signal required by the Wishbone protocol. It indicates that the slave is busy, therefore, the will be repeated until stall is low.

1. Wbs\_err\_o

The error signal required by the Wishbone protocol. TBD – how we use it?

1. Op\_fifo\_wr\_en

This signal is an enable to write opcodes to the Opcode FIFO block. It is active when a new opcode was united and is ready to be stored in the FIFO.

1. Op\_fifo\_data\_in[23..0]

The data signal is the united Opcode to be stored in the Opcode FIFO block.

1. Opcode\_cnt[9..0]

This signal indicates is a counter of opcode bytes.

### FSM for Opcode Unite block

The FSM for Opcode Unite block:

**reset**

**MP1 state**

Opcode\_i(23:16) <= wbs\_data\_i(7:0)

Counter++

**MP2 state**

Opcode\_i(15:8) <= wbs\_data\_i(7:0)

Counter++

**MP3 state**

Opcode\_i(23:16) <= wbs\_data\_i(7:0)

Counter++

Opcode\_fifo\_data\_in <= opcode\_i

**Wbs\_cyc = '1'**

**Wbs\_stb = '1'**

**Wbs\_cyc = '0'**

**Wbs\_cyc = '1'**

**Wbs\_stb = '1'**

**Counter < wbs\_tga\_i**

**Wbs\_cyc = '1'**

**Wbs\_stb = '1'**

**Wbs\_cyc = '1'**

**Wbs\_stb = '1'**

Reminder: The opcode consists of 24 bits, therefore 3 bytes.

Details about the FSM and its states:

1. IDLE state

In the IDLE state, we wait for valid data on the slave Wishbone interface of the block.

We reach this state when:

* After reset.
* When data transmit on the slave Wishbone interface was finished.

1. MP1 state

In the MP1 state, the MSB part of the opcode is assigned in the internal signal.

We reach this state after the IDLE state, when the there is valid data on the slave Wishbone interface.

Counter signal, which counts the received data bytes, is increased by 1.

1. MP2 state

In the MP2 state, the middle part of the opcode is assigned in the internal signal.

We reach this state after the MP1 state, when the there is valid data on the slave Wishbone interface.

Counter signal, which counts the received data bytes, is increased by 1.

1. MP3 state

In the MP3 state, the LSB part of the opcode is assigned in the internal signal.

We reach this state after the MP2 state, when the there is valid data on the slave Wishbone interface.

Counter signal, which counts the received data bytes, is increased by 1.

We assign the output signal opcode\_fifo\_data\_in(23:0) with the value that opcode\_i(23:0) holds.

Note: In the transition from MP3 to IDLE state, when there is no more valid data on the slave Wishbone interface, we will check if the received amount of data bytes equals the amount promised by the Wishbone (AKA - wbs\_tga\_i signal). If not – then it is an error situation and wbs\_error\_o signal will be asserted.

## Opcode FIFO

**Goal:** The main purpose of the Opcode FIFO is to store commands from the Opcode Unite block. The reason we want store the commands before transferring them to the Main RAM is because we don't want to override the current state of the display in the RAM, in case it is the middle of the frame. We will write to the RAM only at start of the frame – when VSYNC of the VESA is active.

**Functionality:**

The Opcode FIFO size is 300 x 24 (rows x bits)

The Opcode FIFO receives the opcodes from the Opcode Unite block. It stores the opcodes until the VSYNC of the VESA is active. Then, the opcodes will be written to the calculated row in the RAM:

RAM\_adr\_wr[8..0] = 20\*x + y

**NOTE:** The FIFO needs to know what the total amount of opcodes is, so that it knows that it finished delivering the data to the RAM. The reason is for debug, therefore this signal will be useful to our simulations.

Also in this block:

1. Calculate the row to write in the RAM (RAM\_adr\_wr[8..0]=). We have 300 rows in total (as the number of (x,y) blocks) ).
2. If com\_type = 0 (remove a symbol) then RAM\_data\_in[0..13]= "0…0"

Else (add a symbol) RAM\_data\_in [0..13]= "com\_add".

1. RAM\_valid ='1'.

**Interface Signals:**

Signals detailed explanation:

Input:

1. Clk\_133

The main clock, to which all the internal logic of the Symbol Generator block is synchronized.

1. Reset

Asynchronous reset of the Symbol Generator block.

1. Op\_fifo\_wr\_en

This signal is an enable to write opcodes to the Opcode FIFO block. It is active when a new opcode was united and is ready to be stored in the FIFO.

1. Op\_fifo\_data\_in[23..0]

Data input to the FIFO. The FIFO receives the opcodes from the Opcode Unite block, and stores them until the VSYNC is active.

1. Op\_fifo\_rd\_start

The read enable of the Opcode FIFO is the VSYNC signal of the VESA. When VSYNC active and Opcode FIFO is ready, only then the FIFO will start.

1. Opcode\_cnt[9..0]

This signal indicates is a counter of opcode bytes.

Output:

1. RAM\_adr\_wr [8..0]

This signal is sent to the RAM. It represents the row to which we want to write to the RAM.

Reminder: each row in RAM represent 1 (x,y) block (total 300 blocks).

1. RAM\_data\_in[13..0]

This signal is sent to the RAM. It represents the data to be send to the RAM (represent the first row in which the symbol is saved in SDRAM).

1. RAM\_wr\_en

This signal is sent to the RAM. It represents the data to RAM is valid, therefore we can write data to the RAM.

1. rd\_mng\_en

This signal is sent to the Read\_Manager block. It is active when t~~he last opcode is sent to the RAM (therefore the RAM is updated to the new desired display)~~ the Opcode FIFO is empty (no new display changes, or the current display changes whee written from the FIFO to the RAM), indicating we can now start generating the new frame (initialize read transaction to the SDRAM).

1. Op\_fifo\_empty

This signal is active when the Opcode FIFO is empty (no rows with data in the FIFO.

1. Op\_fifo\_full
2. This signal is active when the Opcode FIFO is full (no free rows in the FIFO).
3. Op\_fifo\_used[8..0]

This signal indicates the amount of used rows in the FIFO, out of the total size (which is 300 rows). Its value is in the range [ 0, … , 300]

1. Opcode\_fifo\_ready

This signal indicates when all the opcodes, received from the Wishbone, where written to the Opcode FIFO block. Only then, the opcodes are ready to be transferred further.

## RAM

**Goal:** In RAM we save the row in SDRAM, in which the symbol of this block starts. (14 bits)

**Functionality:** The Command FIFO writes to Main RAM. Read\_Manager (RM) reads from RAM.

**Interface Signals:**

Signals detailed explanation:

Input:

1. Clk\_133

The main clock to which all the internal logic of the Symbol Generator block is synchronized.

1. Reset

Asynchronous reset of the Symbol Generator block.

1. RAM\_adr\_in[8..0]

Signals for writing to the RAM

The row number of RAM to which we want to write.

Remark: we need 300 rows for each (x,y) block (therefore we need 2^9 bits)

1. RAM\_wr\_en

Enable signal for writing data to the RAM.

1. RAM\_data\_in[13..0]

Data from the Opcode FIFO to be written to the RAM.

Signals for reading from the RAM

1. RAM\_adr\_rd[8..0]

The row number of the RAM from which we want to read the data.

1. RAM\_rd\_en

Enable signal for reading data from the RAM.

Output:

1. RAM\_data\_out[13..0]: the data read from RAM.
2. RAM\_out\_valid: equals 1 when RAM\_data\_out is valid.

## Read Manager - RM

**Goal:** reading data from the RAM, sending it to SDRAM through WBM , receiving symbols from SDRAM through WBM, saving it in internal FIFOs, sending it to VESA according to certain algorithm.

**Functionality:** The Read Manager block is the controlling "brain" of the Symbol Generator block.

The main functions of this block:

1. Calculating the row in the RAM, from which it manages the read. Including asserting the RAM\_rd\_en signal for the RAM.
2. Receiving the address of the symbol in the SDRAM from the RAM, and calculating the row and column in the SDRAM according to the current video row in the display frame.
3. Managing the toggling between the two FIFOs, using a final state machine. This FSM controls in each state, to which FIFO we write and from which FIFO we read the data. This involves also asserting the control signals of both FIFOs (read and write enable).

**Interface Signals:**

Signals detailed explanation:

Input:

1. Clk\_133

The main clock to which all the internal logic of the Symbol Generator block is synchronized.

1. Reset

Asynchronous reset of the Symbol Generator block. .

1. RAM\_data\_out[0..13]

This signal is sent from the RAM . It represents the data readed from the RAM (represent the first row in which the symbol is saved in SDRAM).

1. RAM\_out\_valid

This signal is sent from the RAM .It is active when the data from the RAM is valid.

1. Wbm\_dat\_i[7..0]

This signal is the data sent from the SDARM.

1. Wbm\_stall\_i

This signal indicates when SDRAM is not ready to transfer data to the Read Manager block.

1. Wbm\_ack\_i

This signal is sent from the SDARM. It is active when WBS acknowledges the read request, and the data from the SDRAM is valid.

1. Wbm\_err\_i

This signal is sent from the SDARM. It is active when an error occurred.

1. req\_in\_trg

This is a signal from the VESA Generator block. It indicates when to start preparing valid data in the Dual Clk FIFO for a req\_lines\_g lines in advance. (In our case it is 1 line in advance).

Output:

1. RAM\_rd\_en

This signal is sent to the RAM. It is an enable signal for reading from the RAM.

1. RAM\_adr\_rd[8..0]

This signal is sent to the RAM. It indicates the row in the RAM from which we want to read.

1. Wbm\_dat\_o[7..0]

This signal is sent to the reg inside SDRAM. It represents the address we want to read from SDRAM.

1. Wbs\_ack\_o – what for, if we send the request to the SDRAM???

This signal is sent to the SDRAM. It is active when data to the SDRAM is valid/ when Wbm\_dat\_i[7..0]\_i is received.

1. Wbm\_add\_o[9..0]

This signal is sent to the SDRAM. It represents the address of the reg inside the Memory block. When the Read Manager block wants to send the address in the SDRAM, then this signal equals 0x2, which is the address of an internal register in the Memory block (implementing read request from the SDRAM in debug mode).

1. Wbm\_tga\_o[9..0]

This signal is sent to the Memory Management block. It represents the reading burst length. It is supposed to be the size of 3 bytes, because the data is the address in the SDRAM and its length is 3 bytes.

1. Wbm\_cyc\_o

This signal is sent to WBM. It indicates the signal cycle required by the Wishbone protocol.

1. Wbm\_stb\_o

This signal is sent to WBM. It indicates the signal strobe required by the Wishbone protocol.

### FSM for Read Manager block

The Read Manager block implements an algorithm for reading data from the SDRAM and transferring it to the Dual Clk FIFO and then to the display.

The intuitive implementation was to read the whole video frame from the SDRAM, store it and then display it. But the HW resources of the board are limited. Therefore we had to find a solution.

In order to reduce the usage of resources in HW, we chose to use two FIFOs (FIFO A and FIFO B), each the size of a video row (640 pixels x 8 bits). The idea was to toggle between them, so that their functionality will change in each stage:

* Reading data from FIFO A and writing data to FIFO B
* Writing data to FIFO A and reading data from FIFO B

The FSM that is implemented in the Read Manager block:

**(req\_in\_trg)**

**AND**

**(last row of the frame)**

**RAM updated**

**Rd\_mng\_en**

**reset**

**(req\_in\_trg)**

**AND**

**(NOT last row of the frame)**

**Finished deliver last row to**

**DC FIFO**

**req\_in\_trg**

**req\_in\_trg**

Details about the FSM and its states:

1. IDLE state

In the IDLE state, both FIFOs are idle – no write and no read to them.

We reach this state when

* After reset.
* When a new video frame is initialized in the VESA Generator (after active state of the VSYNC), but before the RAM was updated with the new changes of the display.
* When the active part of the video frame (AKA: ROI = region of interest) was finished.

1. WRITE A state

In the WRITE A state, FIFO A is in a write mode (video row is written to it) and FIFO B is idle.

We reach this state after the IDLE state, when the RAM was updated with all the new changes to the display.

The data that is written to FIFO A is the first video row of ROI.

1. READ A, WRITE B state

In READ A, WRITE B state, data is read from FIFO A (the next video row to be transferred to the display) and new data is written to FIFO B (the next video row).

We reach this state when VESA Generator requests a new video row and this is NOT the last video row in the frame.

1. WRITE A, READ B state

In WRITE A, READ B state, data is read from FIFO B (the next video row to be transferred to the display) and new data is written to FIFO A (the next video row).

We reach this state when VESA Generator requests a new video row and this is NOT the last video row in the frame.

1. READ B state

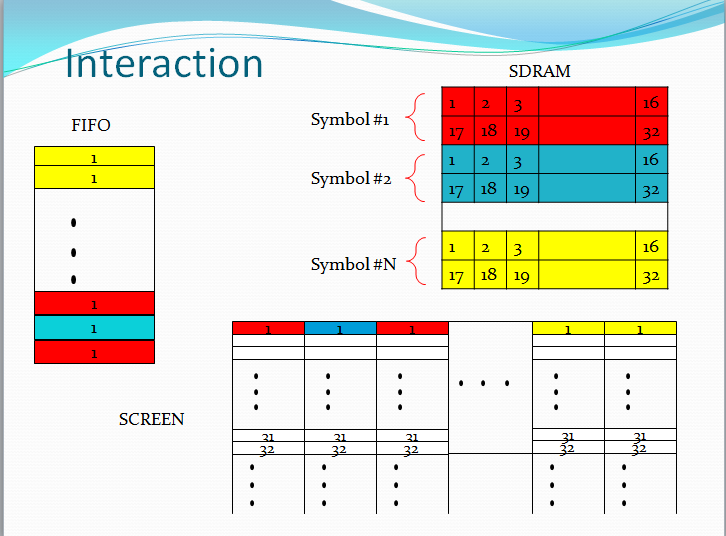
In READ B state, data is read from FIFO B (this is the last row of ROI in the video frame), and FIFO A is idle.

We reach this state when VESA Generator requests a new video row and this is the last video row in the frame.

States " READ A, WRITE B" and "WRITE A, READ B" implement the toggle between the two FIFOs, as explained above.

### Algorithm for Calculating address in Read Manager block

Interaction between symbols in the RAM, SDRAM and on the display (taken from our characterization presentation):



The signals we will use:

**Sym\_row** = the row in terms of symbols, values: 0,…14

**Sym\_col** = the col in terms of symbols, values: 0,…,19

**Row** = the row inside the current symbol, values: 0,…,31

Analogy to:

For **sym\_row**=0 to 14

For **row**=0 to 31

For **sym\_col**=0 to 19

Getting the DATA from the RAM:

For a given **sym\_row** and **sym\_col** : RAM\_ROW <= 20\***sym\_row** + **sym\_col**;

Calculating the address in the SDRAM:

* If **row**=0…15 then we will use the first row of the symbol in the SDRAM.

Row\_SDRAM<= RAM\_DATA\_o&'0';

Col\_SDRAM<= 16\***row**;

Length of the burst is 16 words (each 16 bits)

* If **row**=16…31 then we will use the second row of the symbol in the SDRAM.

Row\_SDRAM<= RAM\_DATA\_o&'1';

Col\_SDRAM<= 16\*(**row**-16);

Length of the burst is 16 words (each 16 bits)

## FIFO A & FIFO B

**Goal:** The purpose of both of FIFO A and FIFO B is to store one video row each, before transferring it to the Dual Clk FIFO.

**Functionality:** The Read Manager block is the controlling "brain" of the Symbol Generator block.

The main functions of this block:

1. Calculating the row in the RAM, from which it manages the read. Including asserting the RAM\_rd\_en signal for the RAM.
2. Receiving the address of the symbol in the SDRAM from the RAM, and calculating the row and column in the SDRAM according to the current video row in the display frame.
3. Managing the toggling between the two FIFOs, using a final state machine. This FSM controls in each state, to which FIFO we write and from which FIFO we read the data. This involves also asserting the control signals of both FIFOs (read and write enable).

**Interface Signals:**