FROM AN ALGORITHM TO A DIGITAL SYSTEM
DEMO SERIES

A VIDEO SERIES ON HIGH LEVEL AND RTL DESIGN USING SYNTHGATE THE TOOL FOR HIGH LEVEL AND RTL DESIGN

DEMO 5:
OTHELLO (REVERSI) STRATEGY GAME DESIGNED IN HLS AND HLS

Begin Demo

FOR MORE INFORMATION GO TO SYNTHEZZA.COM

SYNTHEZZA
HIGH LEVEL & RTL DESIGN

Before watching each video, I highly recommend that you read the introduction to the demo videos series.

In this video, I will demonstrate High level and Register Transfer level design with our tool Synthagate – a product of Synthezza company. As an example, we will discuss the design of a game called Reversi (Othello). It is an abstract strategy board game invented during the Victorian era.

It has remained popular for over a century, regaining popularity in the 1970s when a version of it was re-marketed under the name 'Othello', and again in the 1990s when a computer version of it was included with the Microsoft Windows operating system.

Othello is a two player game on an eight-by-eight square grid with pieces that have two distinct sides. The pieces typically appear coin-like, with a light and dark face, each side representing one player. The goal for each player is to make their colour pieces occupy the majority of pieces on the board at the end of the game, by turning over as many of their opponent's pieces as possible. You can find the rule for this game here.

The difference between Reversi and Othello is minimal. In Reversi, if one player cannot play a piece, the game finishes. In Othello, a player without a move passes, and the other player keeps playing pieces until the first player can make a move again.

Let's begin the design. Fig. 1 presents the working folder before the design. At the beginning, we have one folder Initial. If the designer would like to use a predesigned IP core, he should put it in the second folder *Components*. In this design, I don't have predesigned cores, so I don't have a *Components* folder.

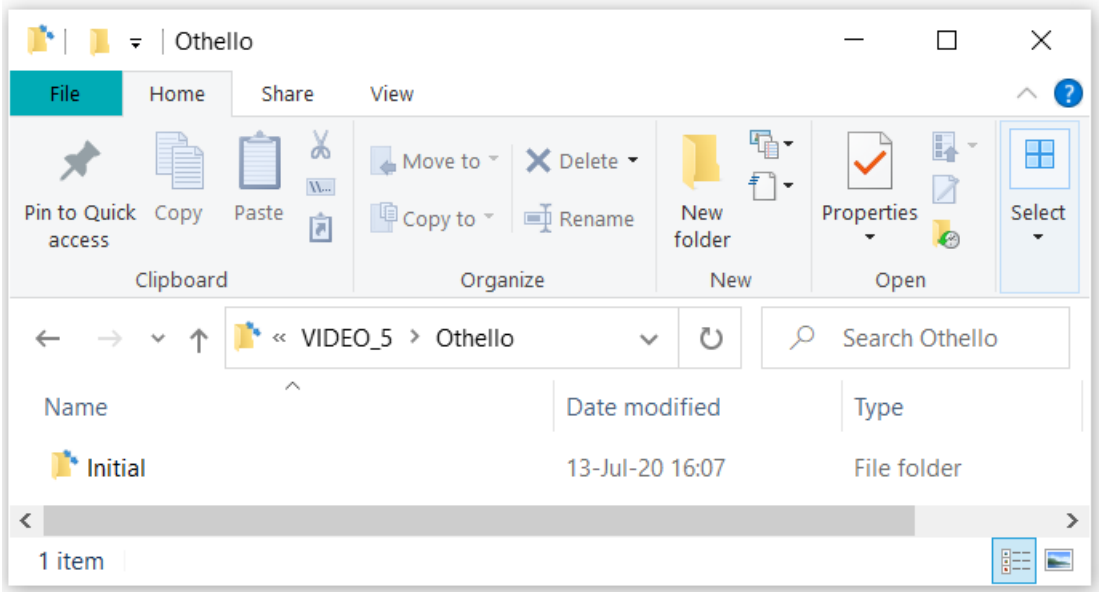

Figure 1. The working folder before the design

Folder *Initial* has only one ASM game (Fig. 2). This ASM was constructed with our GUI ASM Creator. You can find more information about ASM Creator at Synthezza.com, and you can even download it and try it for free. As you see, ASM contains vertex Begin, vertex End, and an operator and conditional vertices. A Boolean expression can be written in the conditional vertex, an operator with one, two, or more microoperations can be written in the operator vertex. There are two kinds of microoperations in ASM. The first kind is a simple microoperation, containing assignment colon-equal ":=". The second kind of microoperation is a component microoperation (we call it *subASM* or a *generalized operator*). Such microoperation does not contain an assign operator colon-equal ":=". In this example, they are colored yellow, but the color is not essential.

These operators are subASMs and will be included in the mother's ASM to get the whole behavior of the digital system or one of its modes if we would like to present them separately. We have five such subASMs in the ASM game.

SubASMs can contain subASMs, and there are no constraints on the number of subASMs and the number of levels of such inclusions. To open each subASM from top to bottom, right click and select "Open".

With generalized operators, we can implement the Top-to-Bottom design. From a design standpoint, complex ASMs aren't necessary. As a rule, they contain about seven-ten vertices. The designer must put generalized operators in the folder *General*, which is a subfolder of folder *Initial*.

You can find all of the ASMs in the folder *Initial* at synthezza.com.

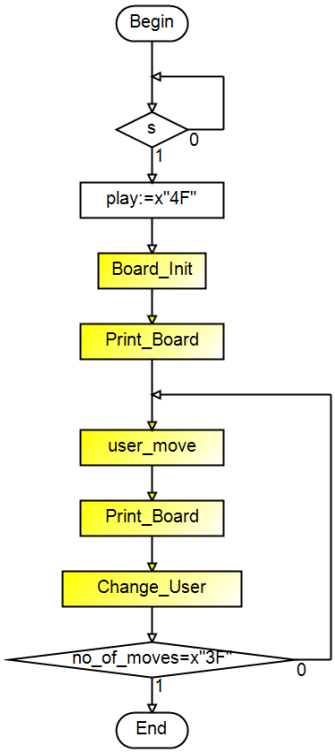

Figure 2. ASM Game constructed with ASM Creator

Thus, before the design, we have *asd*, *gsa*, and *txt* files in folder Initial and its subfolder *General*. Now, that I have explained the content of the working folder before the design, we can begin the design of our game. I am opening our small interface to demonstrate the Synthagate design tool (Fig. 3).

I put the name of the working folder *d:\VIDEO_5\Othello* in the line *Project Folder* of our interface. I checked the box *ASM* (I begin with ASMs from GUI) and didn't check the box *Combine ASMs* (I have only one ASM in this design), checked the box *Include Generalized* (I have several subASMs in folder General) and checked the box *Func Spec* to construct *Functional Specifications*.

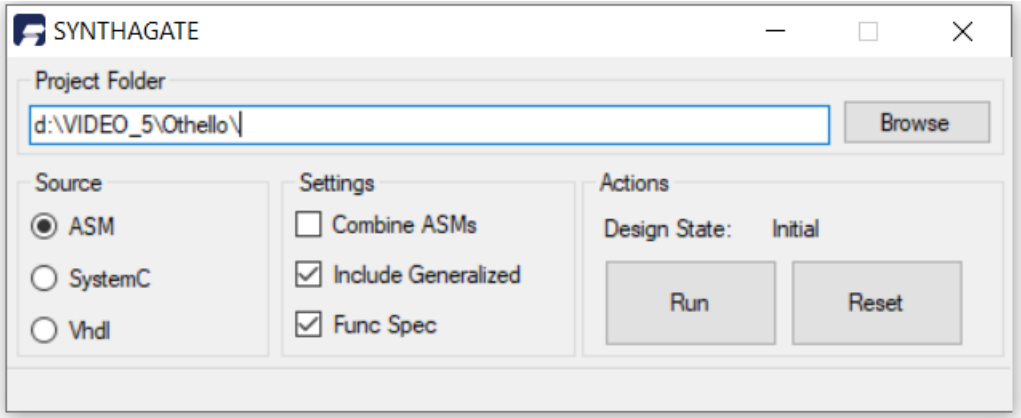

Figure 3. Interface for design

To design the game at the High Level and RTL, I must click Run. And immediately Synthagate suggests that I construct *Functional specifications* in the dialog mode.

In this design, Synthagate asked me only one question. Here I insert the number of words in *bor* (Fig. 4).

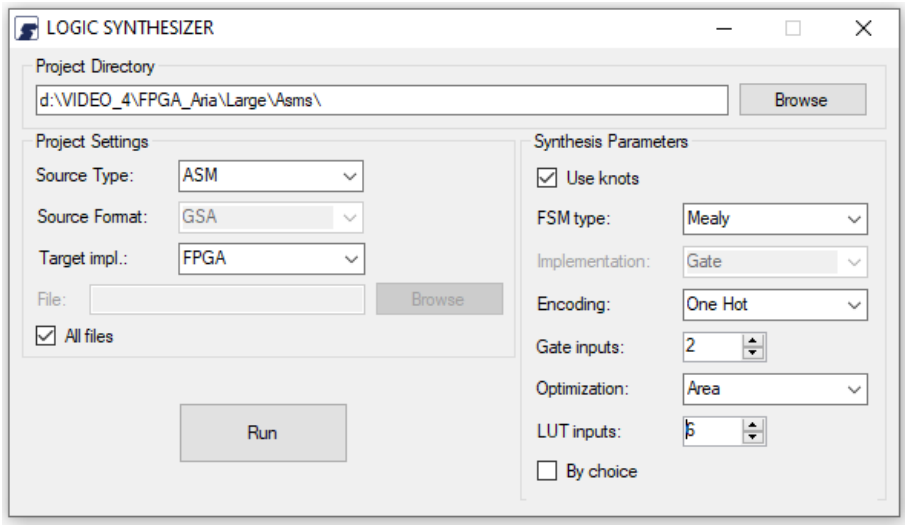

Figure 4. Only one answer in the dialog mode

After I input “64”, the special Synthagate program automatically creates a folder Spec and an *xml* code of the functional specification – file *Funcmi.spec* in this folder (Fig. 5). This file contains specifications of eight ports, 20 signals and one variable. And Synthagate required only one input field.

```
<functional_specifications>
    <declarations>
        <port name="bor_flag" width="1" direction="in" />
        <port name="data_x" width="6" direction="in" />
        <port name="data_y" width="6" direction="in" />
        <port name="ext_adr" width="6" direction="in" />
        <port name="ext_in" width="8" direction="out" />
        <port name="s" width="1" direction="in" />
        <port name="screen" width="1" direction="in" />
        <port name="try_again" width="1" direction="out" />
        <signal name="bac" width="6" />
        <signal name="bor" width="8" size="64" />
        <signal name="br" width="8" />
        <signal name="cd" width="6" />
        <signal name="col_8b" width="6" />
        <signal name="cold" width="6" />
        <signal name="d" width="6" />
        <signal name="i" width="6" />
        <signal name="move_good" width="1" />
        <signal name="no_of_moves" width="8" />
        <signal name="opp" width="8" />
        <signal name="play" width="8" />
        <signal name="rd" width="6" />
        <signal name="rowd" width="6" />
        <signal name="rx" width="6" />
        <signal name="ry" width="6" />
        <signal name="skip" width="1" />
        <signal name="temp" width="6" />
        <signal name="x" width="6" />
        <signal name="y" width="6" />
        <variable name="bor_adr" width="6" />
    </declarations>
    <required_libraries>
        <vhdl>
            <library name="my_package" />
        </vhdl>
    </required_libraries>
</functional_specifications>
```

Figure 5. Functional Specifications *Funcmi.spec*

And in about 4 seconds the design is complete – both at the High Level and at the Register Transfer Level. As a result, we have 13 new folders in the working folder (Fig. 6). Each folder contains one or several stages of the design, together with the complete documentation for these stages.

The result of HLS is presented in folder *HLS*. Synthagate automatically constructs two designs at the same time – the first in VHDL (*Funcmi.vhd*) and the second in System C (*Funcmi.cpp*). File *Funcmi.vhd* is a Finite State Machine (FSM) with 70 states. Its length is 728 lines. This FSM presents our whole design at the High level in VHDL. File *Funcmi.cpp* is a little bit longer – 961 lines with the same 70 states, and it also presents our design at the High level in System C.

After preparing a test bench, the designer can simulate the functional project with any simulation tool. The designer must choose what they would like to use in the simulation at the High level – version in VHDL or version in System C. The same testbench can be used at the last stage of design – at the RTL level. I will return to this simulation a bit later.

Now, let's look at the RTL design. In the folder *RTL*, Synthagate generates VHDL codes for 14 components of the Data path listed in file *Components.vhd*: one *bor*, five *MUXes*, one *ALU* (Arithmetic Logic Units), three *counters*, two *registers*, one *RS flip-flop*, and one *comparator*. At the last step of the Data path design, Synthagate automatically instantiates these components in the *Data path* (file *dp.vhd*) in folder *RTL*. At the bottom of this file, you will find that it contains 378 lines and 46 components instantiated in the Data path. The number of instantiated components is more than the number of VHDL files in folder *RTL* because some components are used several times.

Synthagate automatically creates the RTL code of the *Control unit* (file *Structm.vhd*). Its input signals are feedback from the Data path and some inputs of the digital system. Its outputs are microoperations, with several implemented in the Data path, and several outputs of the designed system. The Control unit is a Finite State Machine with 86 states and 1085 lines.

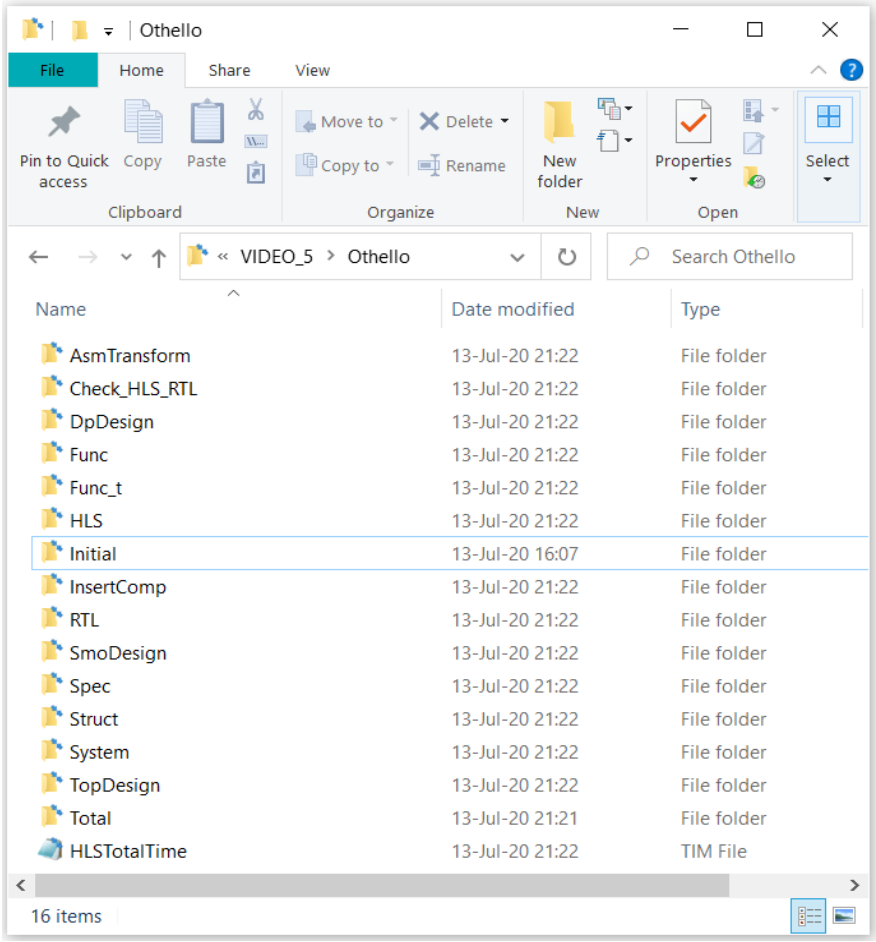

Figure 6. The working folder after the design

At the last stage, Synthagate creates the top-level (file *top.vhd*) automatically combining two components – the Control unit and the Data path. The entity of this file contains inputs and outputs of the designed system. Its architecture has two components – the Control unit (*Structm*) and Data path (*dp*), with signals connecting these components and instantiations of these two components (*u1_fsm* and *u2_dp*).

Now let's talk about the simulation of the game. To begin the simulation, I prepared the folder *SIMULATION* with two subfolders HLS and RTL for the simulation at High level and Register Transfer Level. I am moving this folder into the working folder d:\VIDEO_5\Othello\. The test benches at HLS and RTL are equal. I gave them different names, so you can compare them. The only difference is in the names of the entities, architectures, and components. I want to mention – High-Level synthesis uses the package *my_package.vhd*, but RTL uses the package *components.vhd*. Both these packages were constructed automatically. I won't take your time by showing you this simulation – you can do that yourself with any simulation tools.

Once again, we have implemented the design containing 269 files in about 4 seconds each. And at the beginning, we only had Algorithmic State Machines, and nothing else. No VHDL, no Verilog etc. You

can find more about the design methodology, realized in Synthagate in my books – "Finite State Machines and Algorithmic State Machines" and "High Level Synthesis of Digital Systems" available on Amazon in ebook and paperback.

Thank you.