CPG 5 Coding and Programming

Introduction to Game Programming

Paul Sinnett, London South Bank
University <paul.sinnett@gmail.com>

Table of Contents

Week 5

Memory and addressing

This week's subject areas are:

- Selecting inputs and outputs
- Cycles and the clock
- Memory

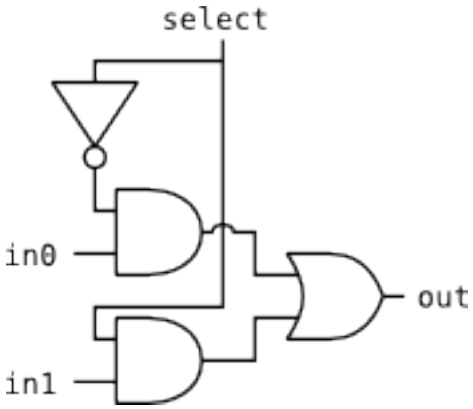
Selecting

So far we've only been considering tests to tell if something is true or not. To be really useful, we want to use the result of those tests to change the state of our game. For example, if we have a test that outputs true when we should score a point, then we want the output to cause our score to increase as a side-effect.

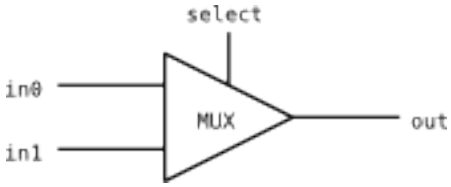
We already know how to add numbers together by arranging Boolean gates in a particular way. If we wanted to select between two possible results based on any Boolean input we could use the following combination of gates:

Figure 1. 2 way 1-bit MUX

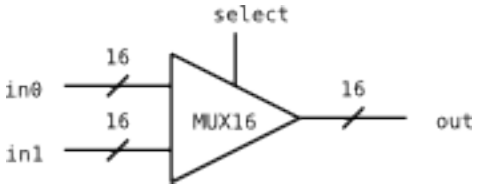
This circuit, sometimes called a multiplexer. It takes two inputs and a select bit. If the select bit is 0, it outputs the input in0, otherwise it outputs the input in1.



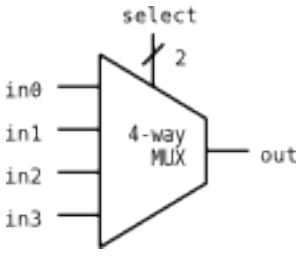
Since this is a generally useful circuit and we'll want to re-use it a lot, there is a standard way to draw it as a black box diagram:



And since it can be duplicated in parallel for each bit of a bus, we can draw it using similar notation for higher bit widths:



We can also duplicate it in series and generate MUX circuits that can select from more inputs.

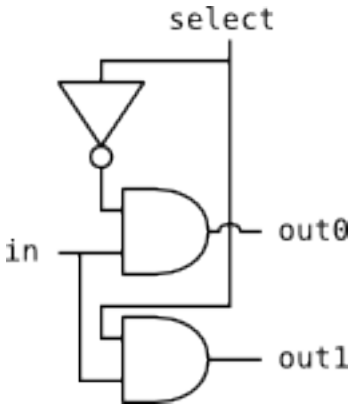


Why do I only need 2 bits to select between 4 inputs?

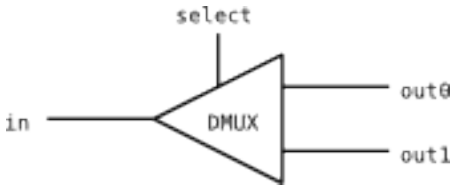
The mirror of this circuit is called the demultiplexer or DMUX:

Figure 2. 2 way 1-bit DMUX

In this circuit one input is directed to one of two different outputs depending on the state of the select input.



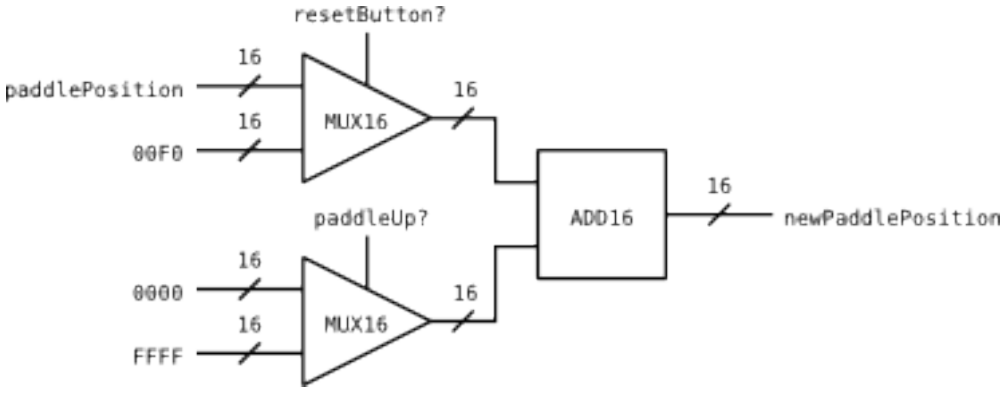
And the corresponding black box version:



By connecting the output of some Boolean circuit, to the select bit of the MUX, we switch the output between the two possible input values.

Figure 3. moving a paddle

This example makes use of 2 MUX circuits. The top one is a reset switch to choose between the current position or the centre of the screen. The bottom one switches if the player presses the *UP* button. It selects between adding 0 or -1 to the other input. The result is the new paddle position for the next update frame.



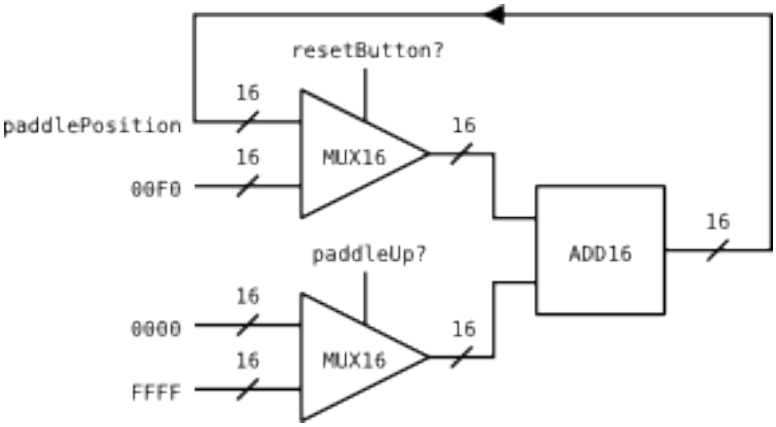
Registers

Given a current state for our game, we now have everything we need to calculate the next state. And we can do this just by re-using circuits that we've already designed. But there's a problem: how do we cycle

to the next stage so that the new state becomes the current state and the circuit can once again calculate the next state?

Figure 4. The feedback problem

Can we just loop the output of the moving paddle example back into the input?



There are two problems here:

The first problem is that it takes a small amount of time for the output states to settle. This is particularly a problem inside the adder because the inputs rely on the outputs of the previous step. To work correctly, all of the input bits must remain constant until the output of the adder has settled.

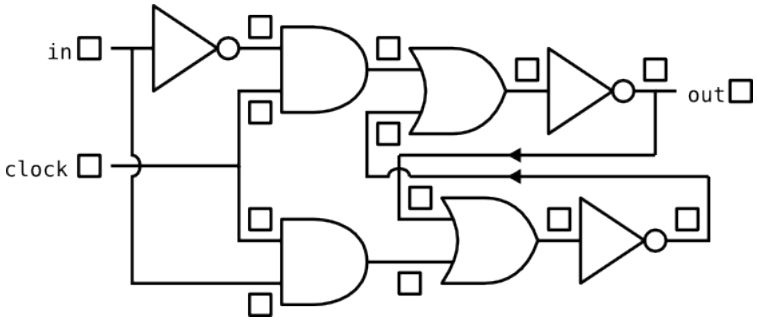
The second problem is due to the speed that these outputs can be calculated. We'd only really need to update the paddle position 50 or 60 times a second. But in simple circuits like this, the output could be updated millions or even billions of times per second.

What we need is a circuit that can hold the value of 1 or more bits constant while we wait for the outputs of our calculations to settle. And optionally take on the new value when we are ready to cycle to the next game state. This circuit is known as a register.

To create a circuit that can hold a value constant, we need to look at a new kind of arrangement of gates. Until now, the flow of all the diagrams we've looked at has been one way. Now we need to introduce a feedback loop.

Figure 5. The DFF (data flip-flop)

This circuit will change its state only when the clock input becomes true. At that point the output becomes the input. When the clock input becomes false, the output remains constants regardless of what happens to the input. This circuit is sometimes called a NAND latch.



Convince yourself that the value should hold while the clock input is false by trying different combinations in the boxes. Note that you won't be able to determine the initial state while the clock input remains false, so begin by setting it to true.

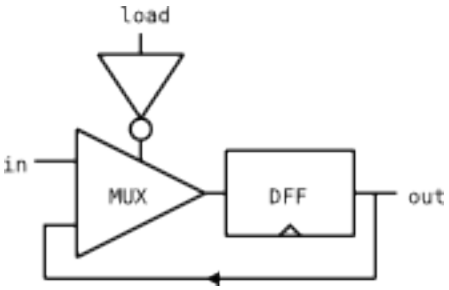
In modern computers, the clock frequency ticks true and false billions of times per second. One billion ticks per second is one Gigahertz.

The black box diagram for a DFF includes a small tick on the bottom of the box. This is indicates the clock signal.

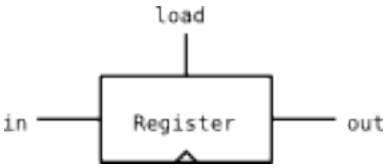
We can use a DFF to create a register circuit. To match the requirements listed above, all we need is a switch to load in a new value or keep the old one. We can do this with a 2 way MUX:

Figure 6. The register

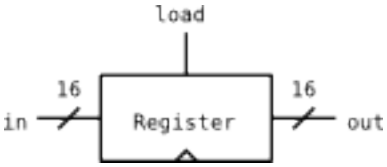
The register is a DFF connected to itself. The MUX selector and load input determine whether the DFF will retain its current value or take a new value at the next tick of the clock.



The black box drawing of a register is the same as a DFF but with the addition of a load input:



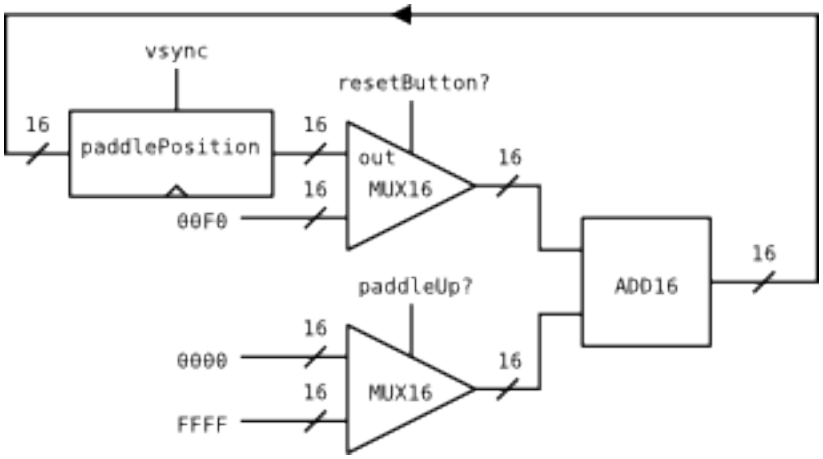
This circuit can be duplicated in parallel to provide a register for any bit width we want. Here's how we would draw out a 16 bit register:



With the addition of a memory register we can now solve the feedback problem above:

Figure 7. A cycling update

The output of the circuit feeds back into the input via a register. This keeps the circuit stable since changes happen only when the clock ticks. If we set the register's load input to a suitable signal (in this example the vertical refresh of the monitor) we can limit the speed of movement to a reasonable amount.



If the screen is 480 pixels high and the vsync is 50Hz, how long would it take for the paddle to travel from top to bottom?

The Brown Box

A good example of an early video game built using these kinds of circuits is Ralph Baer's Brown Box.

Article on the Brown Box console [http://www.humansinvent.com/#!/4106/game-changing-prototypes-3-the-brown-box/]

In the illustrations to the article you can see just how dense the wiring is.

Memory

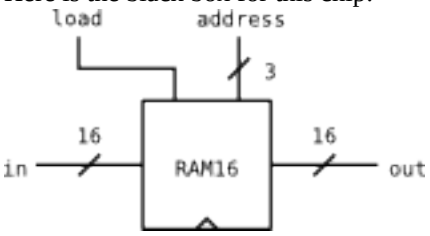
By combining registers and selectors we can create a circuit that selects from a number of internal registers. This circuit is what we call a memory circuit, or more usually, a memory chip.

The number of registers inside such a chip gives us the size of the memory. For example, a 16K memory chip might contain 16,384 8 bit registers or 8,192 16 bit registers, and so on. The selection of a specific register to read from or write to within this chip is known as the address.

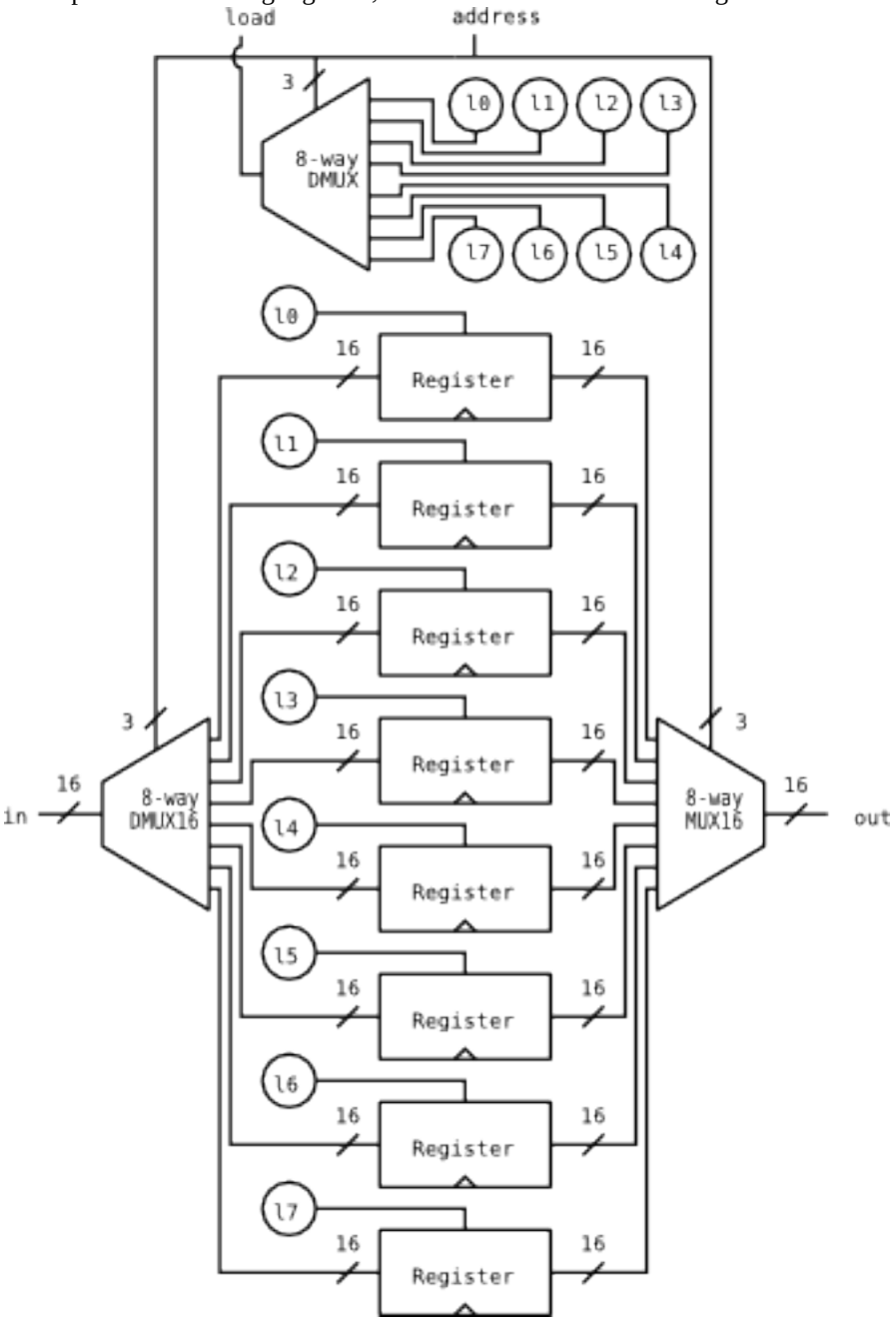
Figure 8. 16 bit RAM16 chip

This chip takes as input one 16 bit value, a load bit, and a 3 bit address. As output it returns the value of the register at the address given. If the load bit is set, the contents of the register at the given address are updated at the next clock pulse.

Here is the black box for this chip:



An implementation using registers, MUX and DMUX selectors might look something like this:



Exercise

For this week's exercise I'd like you to draw out a circuit diagram to encode a rule of your game. Pick a rule which modifies the state of the game if the rule is true. Use the same diagram conventions as in these notes. And make sure your diagram includes a feedback through a register.

You may use any software you like to create your diagrams, or draw them by hand on paper. But you must submit the result as either a PDF or JPG format file.

Submit your files as usual to source control before Sunday. And don't forget to include your work log with your submission.