U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

EEC0055
Projecto de Sistemas Digitais

Laboratory 3: All-digital FM stereo modulator

João Beleza, Pedro Costa
{up201402831, up201402793}@fe.up.pt

January 14, 2019

1 System overview

This report provides information on an all-digital FM stereo modulator developed as a final project for Projecto de Sistemas Digitais.

A general overview of the project and its sub-modules can be seen in Figure 1.

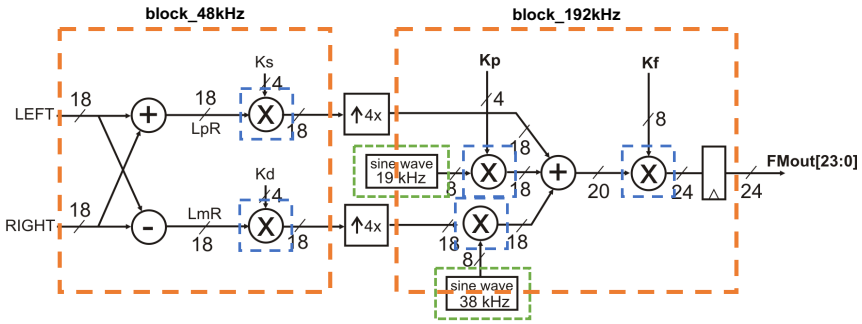

Figure 1: Block diagram overview of the project. Implemented macro blocks are highlighted in orange; DDS blocks are seen in green; improved multiplication blocks are seen in blue.

The implementation of this project was wrapped under a single module named `my_fm_module`, which was split into the following blocks:

- `block_48kHz`: corresponds to the leftmost block, which receives the stereo input at a sample rate of 48 kHz. Its outputs will feed each of the interpolators.
- `interpol_4x`: interpolate the signal for further processing. The blocks used were provided as IP blocks.
- `block_192kHz`: corresponds to the rightmost block, which processes the signal at a rate of 192 kHz. It outputs the final FM signal.

Besides this, two additional modules were implemented:

- `dds`: implements each of the sine wave generators. Implementation details can be read in section 2.
- `seqmultNM_sat`: implements a wrapper of the original `seqmultNM` IP block. It implements some of the control logic detailed in section 3, as well as a comprehensive mechanism that allows for rescaling the number and for saturating the output, thus avoiding overflow.

All sums and shifting operations were performed using standard Verilog code.

Furthermore, some optimisations were performed throughout the implementation of the project. These are described under section 4.

2 DDS module

In this section, the DDS (Direct Digital Synthesiser) modules implemented in the project will be discussed.

Below follows the implementation used in the final version of the project:

```
* Parameter definitions are used for
  - NBITS_PHASE: # of bits of phase
  - NBITS_PHASE_FRAC: # of bits of fractionary part of phase
  - NSAMPLES_LUT: # of samples used in the LUT (Lookup Table)
  - HEXVAL: string containing the path for the values to be inserted in
     the LUT
* Values are uploaded from the file to the LUT using $readmemh()
* With posedge clock, if reset is not active:
  - if enableclk input is active, then phase must be updated
  - outsine output is updated in accordance with the latest value of
     phase
```

Listing 1: Pseudo-code for final DDS implementation

Besides this, a second DDS implementation was developed with the purpose of reducing the final circuit size. To achieve this, a module taking advantage of the periodicity properties of the wave, shown in Figure 2, was implemented.

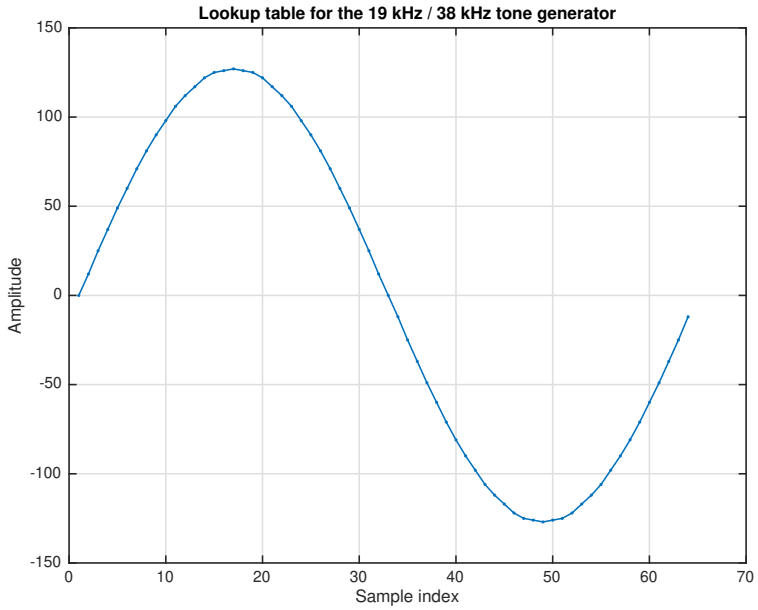

Figure 2: Sine wave waveform and LUT values

Because the number of samples of the LUT is a power of two, it is possible to infer the complete signal by only storing a fraction of the signal. This allowed for a reduction of the size of the LUT by 75%. Upon further testing, it was verified that this implementation

provides for area optimisations. Among others, the number of block RAM/FIFO was reduced from 1 to 0 (as seen in Table 1).

The module was implemented as follows:

```
* Parameter definitions are used for
  - NBITS_PHASE: # of bits of phase
  - NBITS_PHASE_FRAC: # of bits of fractionary part of phase
  - NBITS_OUTPUT: # of bits in the output
  - NSAMPLES_LUT: # of samples used in the LUT (Lookup Table)
  - HEXVAL: string containing the path for the values to be inserted in
     the LUT
* Values are uploaded from the file to the LUT using $readmemh()
* sineLUT_i variable is assigned to translate the previously used phase
   logic to a logic that supports a smaller LUT

* With posedge clock, if reset is not active:
  - if enableclk input is active, then phase must be updated
  - if value of phase equals NSAMPLES_LUT/4, sine output equals to the
     maximum representation value
  - if value of phase equals 3*NSAMPLES_LUT/4, sine output equals to the
     minimum representation value
  - else, if phase[NBITS_PHASE-1] (the MSB for phase representation is
     enabled), then values should be output from the LUT as negative
  - else, values should be output as is from the LUT
```

Listing 2: Pseudo-code for an optimised implementation of the DDS

3 Control path

In this section the control path will be explained, followed by the implementation of both `block_48kHz` and `block_192kHz` blocks. These are an application of that same algorithm.

The multiplication block is essential for the correct functioning of the system. As such, the control path was designed around its requirements, as detailed below.

```
* Start phase:
  - The inputs are continually updated in accordance with each
     multiplier
  - The start flag of the multiplier is enabled
* Multiplier phase:
  - The control flag is enabled when the ready flag is low
* Final phase:
  - Results are signalled as ready when both control and ready flags are
     enabled
  - Upon acknowledging the valid results, the control flag is set to low
```

Listing 3: Description of the control path

3.1 block_48kHz

For this block, both inputs are being added and subtracted continuously by means of an *assign* condition. Any possible overflow is avoided by discarding the LSB as a result of a previous sum which preserves any overflow information.

Due to requirements of the `seqmultNM` blocks, both `Ks` and `Kd` gains are extended by a signal bit, so as to be interpreted as signed (and positive) variables.

The multiplier is enabled using the signal `clocken_48`, which is active during a single clock cycle, with a periodicity of 48 kHz. After this, the strategy in Listing 3 is used, and its output is directed to the inputs of the `interpol4x` blocks. Using the `seqmultNM_sat` described under section 4, the outputs of both multipliers are shifted by 3 bits to the right.

3.2 block_192kHz

For this block, a similar implementation is performed. The gains `Kp` and `Kf` are extended by a signal bit. The output of both `dds` blocks is placed in a 8-bit register.

The multipliers at the left of the sum block (see Figure 1) are enabled using the signal `clocken_192`, which is active during a single clock cycle, with a periodicity of 192 kHz. Besides this, a condition is added, which signals that all calculations are finished within the block. For the multiplication with the 38 kHz sine wave, the result is affected by an 8-bit right shift. For the multiplication with the 19 kHz sine wave, the result is affected by an 6-bit left shift. Both outputs are rescaled using the `seqmultNM_sat` block.

A single *assign* statement is used for the summation of all three values before the rightmost multiplication. When both multiplications before this sumation finishes, as described under Listing 3, the results are acknowledged. This disables the flags of the multipliers on the left of the summation, and enables the rightmost multiplication. Its output already counts for a 4-bit shift. A control flag is enabled, which states that all calculations are finished, and its values are loaded into a registered output with a periodicity of 192 kHz.

4 Further optimisations

4.1 Architectural optimisations

In order to optimise the circuit, a minimum area target was defined.
To achieve this, some strategies were followed, as is explained below.

Coding

Below are listed the performed optimisations:

- Definition of `seqmultNM` blocks with `N` (multiplier) having the most bits and `M` (multiplicand) having the least bits. This increases the number of cycles needed for computation (but still manages to meet timing constraints), but allows for decreases in area
- Elimination of all duplicate `reg` and `wire` instantiations

Further optimisation was also attempted. However, some measures didn't translate into additional optimisation gains. The attempted procedures are listed below:

- Condensation of all division and multiplication scenarios in `seqmultNM_sat` by using a *wire* assignment before comparing the final value
- Utilisation of `else ... if` statements when in the presence of mutually exclusive statements (e.g. `x` versus `!x`)
- Utilisation of registers to save all previous inputs from multipliers, in order to lower power consumption in case the results remained unaltered (removed due to increase in area size)
- Utilisation of two-step sum in `block_192kHz` (i.e. by performing two sums with two operands instead of a single sum with three operators

Synthesis

After validation of the final design, synthesis optimisation was achieved by means of editing the `Optimisation Goal` and `Optimisation Effort` parameters under the `Synthesis - XST > Process Properties` options.

Maximum optimisation was achieved using the values listed below. The resulting device utilisation statistics are presented in Table 1.

Optimisation Goal	Speed
Optimisation Effort	Normal

4.2 Resource occupancy

The results for the resource occupancy of the device upon synthesis follow below.

Logic utilisation	Used	Available	Utilisation
Number of Slice Registers	1732	54576	3%
Number of Slice LUTs	1319	27288	5%
Number of fully used LUT-FF pairs	898	1778	50%
Number of bonded IOBs	78	218	35%
Number of Block RAM/FIFO	0	116	0%

Table 1: Device utilisation results for the final optimised synthesis

4.3 Additional changes

Besides this, a change was performed to the FM stereo data generation data generation script (`fmstereo_gensimdata.m`). An imprecision was detected in lines 135 and 137, in which the generated data, when saturating to a negative value, would exceed the representation range of the hardware, a wrong value. The code was changed so as to saturate to the most negative value possible to represent by its number of bits.