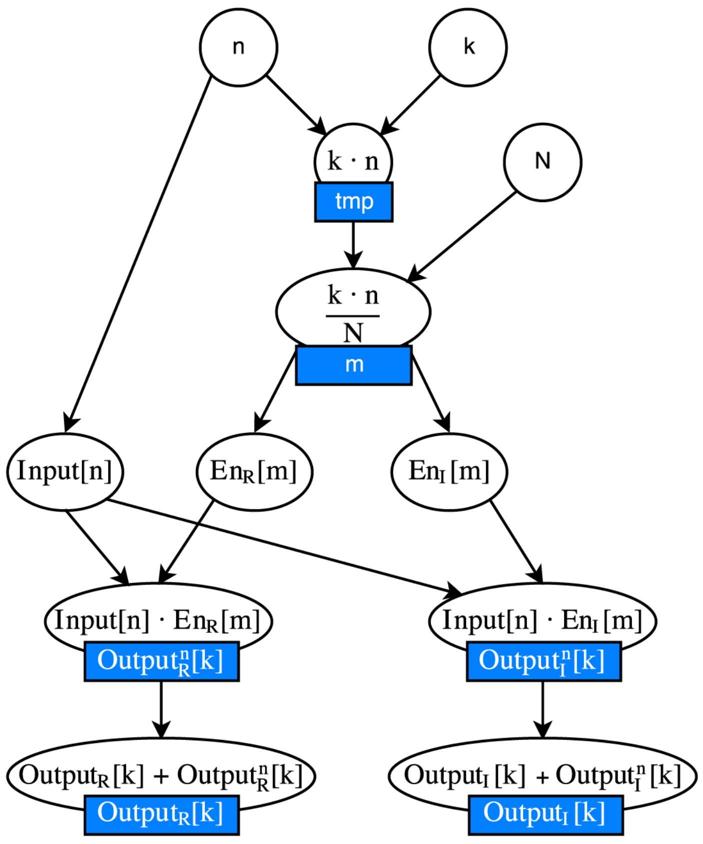
dsPIC33: Exercise

Pierre Gerard, Julian Schrembi, Francois Schiltz

April 6, 2016

1 Question 1



2 Question 2

2.1 Computation of the value m

```
INT16U unsigned16temp;
unsigned16temp = (k * n);
```

`m = (INT8U) (unsigned16temp / (INT16U) N);`

If we directly do the computation $m = k*n/N$ we would get a wrong result. In fact the multiplication would be too big and some of the result could overflow and be wrong. To solve that problem, we use a intermediate variable of 16bit to store the result of the multiplication. We then do the division and then cast it back to 8 bits. The cast should be correct because the result of the division (m) should stands on 8 bits if the inputs are correct.

2.2 Computation of the real part

`signed16temp_r = (INT16S)input[n] * (INT16S) en_r[m];`
First we cast the two 8 bits fixed point integer to two 16 bits fixed point integer to avoid sign trouble and then do the multiplication as shown in the $example2.c$. We put the result in a 16bits integer so the result would be correct.
`signed8temp = (signed16temp_r+128) >> 8;`
Then we are required to put the result in a 8 bits register, so to get better result we first round it and then do the shift to only keep the most significant bits.
`signed16temp_r = (INT16S) signed8temp + (INT16S) output_r[k];`
At last we add the element of the sum to a temporary 16 bits register. We use a 16 bits temporary variable because the addition of two number on 8 bits could overflow.

2.3 Computation of the imaginary part

Actually it is the same as the real part with other variables.

3 Question 3

3.1 Computation of the value m

`unsigned16temp = (k * n);`
Since we do INT16 = INT8 * INT8 no overflow will occur. Let's take a look at a borderline case : $1111\ 1111 * 1111\ 1111 = 1111\ 1110\ 0000\ 0001$. There is no information lost so the accuracy is maximum.
`m = (INT8U) (unsigned16temp / (INT16U) N);`
If the input are in there correct range, no overflow will occur because m must be between 0 and 127 included, so it is ok to cast it to a 8 bit integer. Some precision can be lost during the division since the result is a integer all the time. Plus it loose the decimal, so for instance 1.9 will be 1 if interpreted as an integer instead of 2 for the nearest integer. Let's take a look at another (non-trivial) borderline case : $0000\ 0001 * 0000\ 0001\ /\ N = 0000\ 0000\ 0000\ 0001/128 \Leftrightarrow 0000\ 0000\ 0000\ 0001 \gg 8 = 0000\ 0000$.

3.2 Computation of the real part

`signed16temp_r = (INT16S)input[n] * (INT16S) en_r[m];`
The two 8 bits fixed point integer are cast to two 16 bits fixed point integer so the sign will be correct. Then since the number stands on 8 bits, it's similar to above for the justification that no overflow happen. Borderline case : $1111\ 1111*$

1111 1111 = 1111 1110 0000 0001. There is no information lost so the accuracy is maximum.

```
signed8temp = (signed16temp_r+128) >> 8;
```

Here no overflow could happen.

```
signed16temp_r = (INT16S) signed8temp + (INT16S) output_r[k];
```

The two 8 bits fixed point integer are cast to two 16 bits fixed point integer so the sign will be correct. Since we put the result of the addition of two number standing on 8 bits in a 16 bits variable no overflow can happen and the accuracy is maximum. Borderline case : 1111 1111 + 1111 1111 = 0000 0001 1111 1110

3.3 Computation of the imaginary part

Actually it is the same as the real part with other variables.

3.4 Adding value to output

```
output_r[k] = (INT8S) (signed16temp_r >> 7);
output_i[k] = (INT8S) (signed16temp_i >> 7);
```

4 Question 4

The more the frequency grows, the nearest the sample are. So that mean that two sample will have almost the same value. From a certain point, we might see that the sample will have the same value.