```
# -----------------------------------------------------------------------------
# File                              : MAP1.arx
# Description   :
# Author                            :
# Creation date:
# -----------------------------------------------------------------------------
# $Rev: 1$
# $Author: $
# $Date: $
# $Log$
# -----------------------------------------------------------------------------


component top
        # declare first fixed-point parameters for data and coefficients
        wl_data: generic integer = 8
        iwl_data: generic integer = 5
        wl_coef: generic integer = 8
        iwl_coef: generic integer = 1

        # now declare the data types for data and coefficients
        T_data: generic type = signed(wl_data, iwl_data, wrap, round)
        T_coef: generic type = signed(wl_coef, iwl_coef, sat, round)

        # now declare the IO
        data_in: in T_data
        data_out: out T_data

# SG: unfortunately, declaring the filter coefficients as constants
# generates the wrong code; the workaround consists of declaring them
# as variables and assigning them a constant value.

# constant
#       # the filter coefficients
#       b2: T_coef = 0.449067766265545
#       b1: T_coef = -0.803316855076157
#       b0: T_coef = 0.449067766265545
#       a2: T_coef = -0.387641686503134
#       a1: T_coef = 0.519937751601787

type
        # the intermediate data type after multiplication
        T_mult: signed(wl_data+wl_coef, iwl_data+iwl_coef)

        # the schedule requires 7 clock cycles
        # the state type
        T_state: enum(cycle0, cycle1, cycle2, cycle3, cycle4, cycle5, cycle6)

register
        # the registers in the design
        # Input & output
        i1: T_data = 0
        o1: T_data = 0

        # registers
        r1: T_mult = 0
        r2: T_mult = 0
        d1: T_mult = 0
        d2: T_mult = 0

        # the counter that counts cycles on behalf of control
        state: T_state = T_state.cycle0

variable
        # multiplier 1 inputs and output
        m_in_l: T_coef
        m_in_r: T_data
```

```
        m_out:  T_mult

        # multiplier 2 inputs and output
        m2_in_l: T_coef
        m2_in_r: T_data
        m2_out: T_mult

        # adder input and output
        a_in_l: T_mult
        a_in_r: T_mult
        a_out:  T_data

        # the coefficients (see remark above)
        b2: T_coef
        b1: T_coef
        b0: T_coef
        a2: T_coef
        a1: T_coef

begin
        # assign the coefficients a value
        b2 = 0.449067766265545
        b1 = -0.803316855076157
        b0 = 0.449067766265545
        a2 = -0.387641686503134
        a1 = 0.519937751601787

        # connect multiplier inputs
        # make sure that the inputs are stable during two clock cycles
        case state
                when T_state.cycle0
                        # m2 cycle 1
                        m_in_l = a1
                        m_in_r = d1
                        # m4 cycle 1
                        m2_in_l = a2
                        m2_in_r = d2

                when T_state.cycle1
                        # m2 cycle 2
                        m_in_l = a1
                        m_in_r = d1
                        # m4 cycle 2
                        m2_in_l = a2
                        m2_in_r = d2

                when T_state.cycle2
                        # m3 cycle 1
                        m_in_l = b1
                        m_in_r = d1
                        # m5 cycle 1
                        m2_in_l = b2
                        m2_in_r = d2

                when T_state.cycle3
                        # m3 cycle 2
                        m_in_l = b1
                        m_in_r = d1
                        # m5 cycle 2
                        m2_in_l = b2
                        m2_in_r = d2

                when T_state.cycle4
                        # m1 cycle 1
                        m2_in_l = b0
                        m2_in_r = d1
                when T_state.cycle5
                        # m1 cycle 2
                        m2_in_l = b0
                        m2_in_r = d1

                when T_state.cycle6
                        # Nothing here
        end

        # connect adder inputs
        # they need to be stable for a single clock cycle

        case state
                when T_state.cycle0
                        # Nothing here

                when T_state.cycle1
                        # Nothing here

                when T_state.cycle2
                        # p3
                        a_in_l = r1
                        a_in_r = r2

                when T_state.cycle3
                        # p1
                        a_in_l = i1
                        a_in_r = r2

                when T_state.cycle4
                        # p4
                        a_in_l = r1
                        a_in_r = r2

                when T_state.cycle5
                        # Nothing here

                when T_state.cycle6
                        # p2
                        a_in_l = r1
                        a_in_r = r2
        end

        # arithmetic
        m_out = m_in_l * m_in_r
        m2_out = m2_in_l * m2_in_r
        a_out = a_in_l + a_in_r

        # new register values
        # specify only content updates; registers preserving their values do
        # not need to be specified
        case state
                when T_state.cycle0
                        state = T_state.cycle1
                        i1 = data_in

                when T_state.cycle1
                        state = T_state.cycle2
                        # m2
                        r1 = m_out
                        # m4
                        r2 = m2_out

                when T_state.cycle2
                        state = T_state.cycle3
                        # p3
                        r2 = a_out
```

```

                when T_state.cycle3
                        state = T_state.cycle4
                        # m3
                        r1 = m_out
                        # m5
                        r2 = m2_out
                        # Shift delay
                        d2 = d1
                        # p1
                        d1 = a_out

                when T_state.cycle4
                        state = T_state.cycle5
                        # p4
                        r2 = a_out

                when T_state.cycle5
                        state = T_state.cycle6
                        # m1
                        r1 = m2_out

                when T_state.cycle6
                        # back to initial state
                        state = T_state.cycle0
                        o1              = a_out
        end

        # wire output
        data_out = o1
end
```