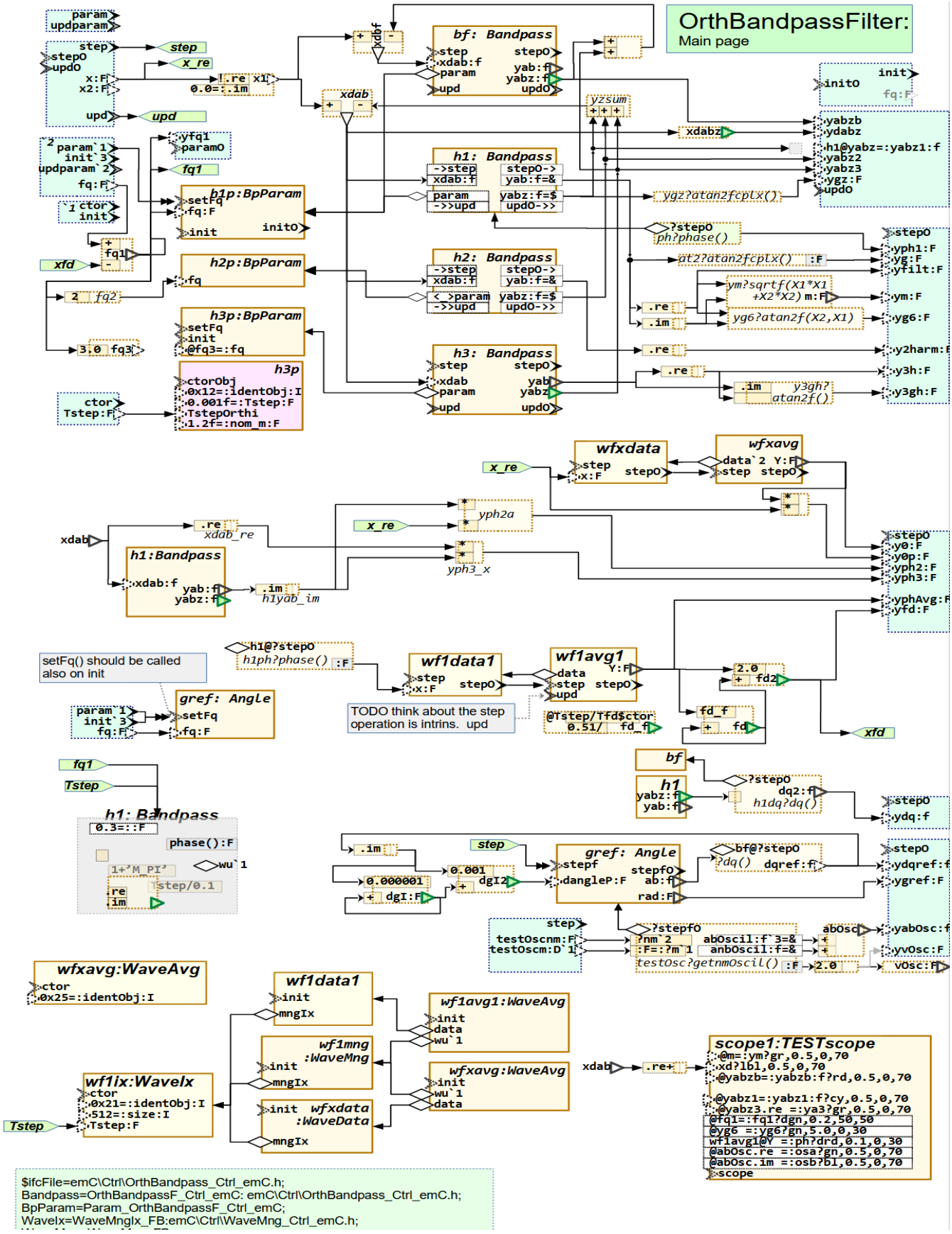
OFB
Object oriented Function Block Graphic in LibreOffice draw
-
with evaluation and code generation

Dr. Hartmut Schorrig
www.vishia.org
2025-03-12

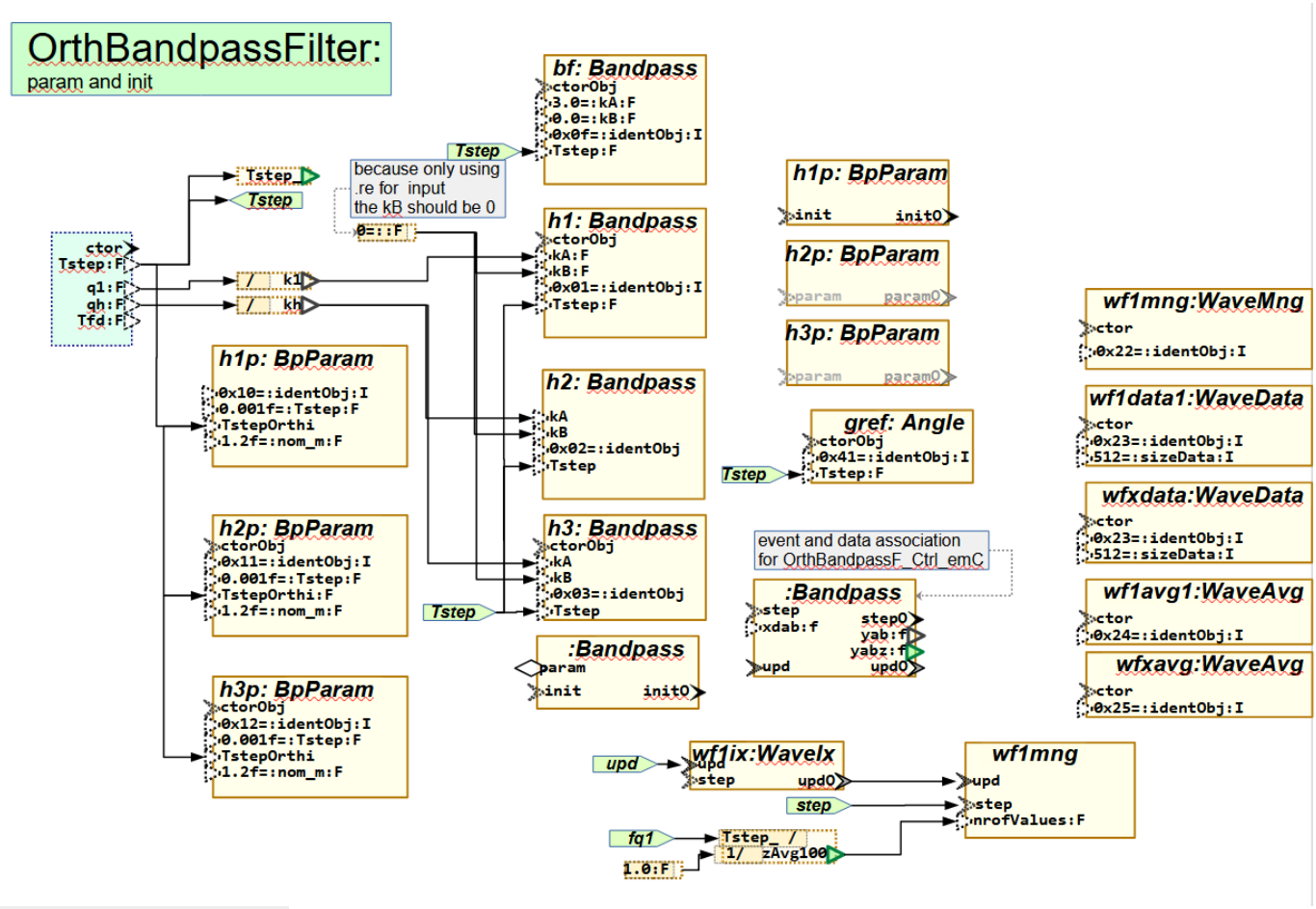
Table of Contents

1 A filter module uses (simple) legacy code in C.....2
2 Module and test PID-Ctrl with different data types.....4
3 Slices and Arrays.....6
4 Conditional execution.....8
5 State machines.....10

1 A filter module uses (simple) legacy code in C



OrthBandpassFilter-1.png

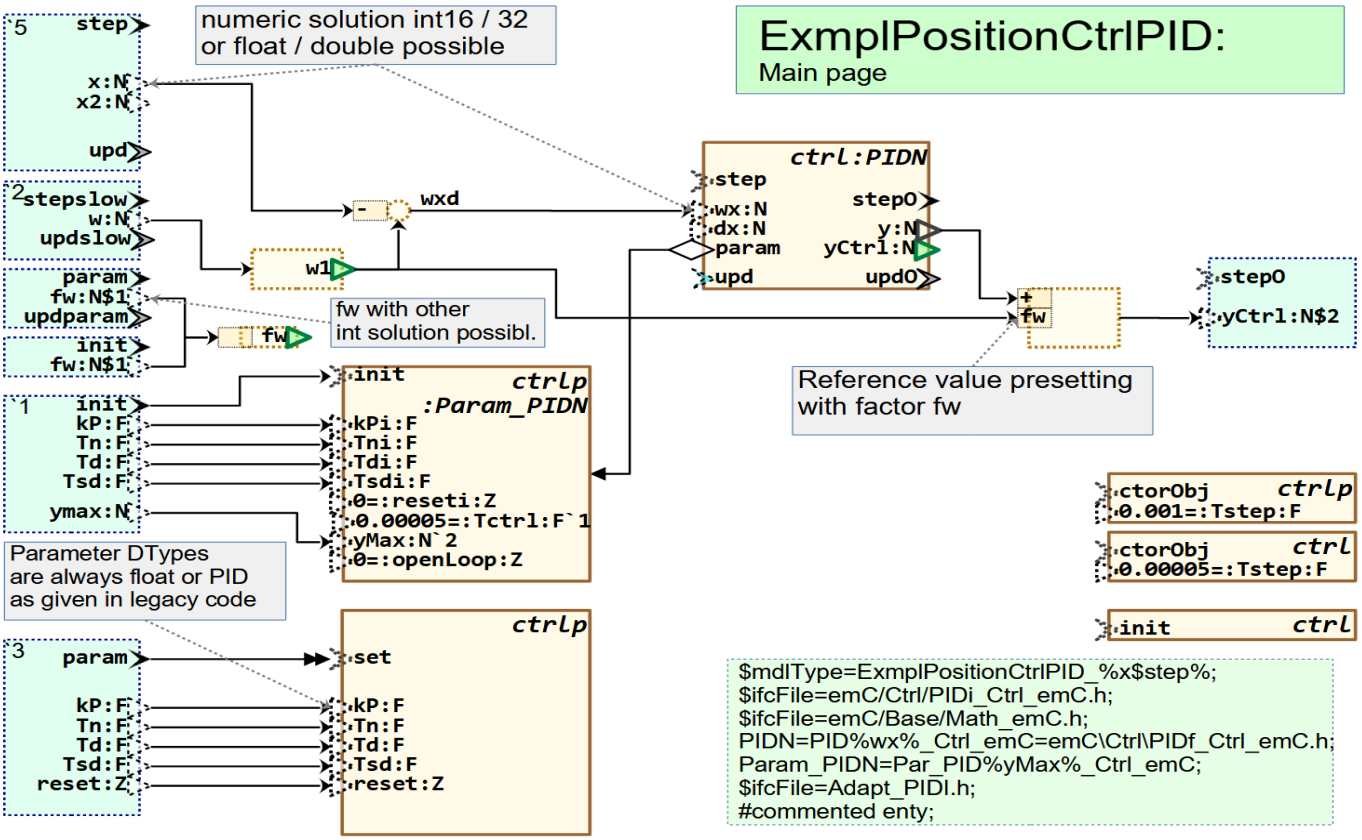


OrthBandpassFilter-2.png

Code Snippet in `cpp/genSrc/OrthBandpassFilter.c`:

```
void step_OrthBandpassFilter ( OrthBandpassFilter_s* thiz, float x, float x2 ) {
  float_complex x1;  // locale variable for x1_X.prep dtype:0`2:f otx: defineLocalVar_FBex
  float_complex dqref;  // locale variable for dqref_X.dq dtype:0`2:f otx: defineLocalVar_
  thiz->mEvout_step = 0;  // set evout bits to 0, will be set maybe conditional otx: GenCod
  // following statements from exec: prcEvchainOperation(...)
  stepf_Angle_abgmf_Ctrl_emC(&thiz->gref, thiz->dgI2_z);  //$module_OrthBandpassFilter.step
  step_WaveMng_FB(&thiz->wf1mng, thiz->zAvg100_z);  //$module_OrthBandpassFilter.step --> w
  step_WaveData_FB(&thiz->wfxdata, x);  //$module_OrthBandpassFilter.step --> wfxdata.step
  step_AvgWave_FB(&thiz->wfxavg);  //wfxdata.stepO --> wfxavg.step  otx:callFB_evin
  x1.im = (0.0);  //step --> x1_X.prep  genExprOut(...) in otx: setVar_FBexpr
  x1.re = (x);  //step --> x1_X.prep  genExprOut(...) in otx: setVar_FBexpr
  thiz->xdab.re = (x1.re - (thiz->h1.yabz.re + thiz->h3.yabz.re + thiz->h2.yabz.re));
  thiz->xdab.im = (x1.im - (thiz->h1.yabz.im + thiz->h3.yabz.im + thiz->h2.yabz.im));  //x1
  step_OrthBandpassF_Ctrl_emC(&thiz->h1, thiz->xdab);  //xdab_X.prepO --> h1.step  otx:call
  step_OrthBandpassF_Ctrl_emC(&thiz->h2, thiz->xdab);  //xdab_X.prepO --> h2.step  otx:call
.....
  //Module outputs due to the event stepO:  otx: evMldOutStart
  thiz->mEvout_step |= MASK_step_stepO;  // set the output event bit otx: evMldOutStart
  thiz->yph1 = phase_OrthBandpassF_Ctrl_emC(&thiz->h1);    // otx: setMdlOutScalar
  thiz->yg = atan2fcplx(thiz->h1.yab);    // otx: setMdlOutScalar
  thiz->yfilt = (thiz->h1.yab.re);    // otx: setMdlOutScalar
....
  thiz->yph3 = ((thiz->xdab.re) * (thiz->h1.yab.im));    // otx: setMdlOutScalar
  thiz->yphAvg = thiz->wf1avg1.Y;    // otx: setMdlOutScalar
  thiz->yfd = thiz->fd2_z;    // otx: setMdlOutScalar
  thiz->ydq = thiz->dq2;    // otx: setMdlOutStructCpy
  thiz->ydqref = dqref;    // otx: setMdlOutStructCpy
  thiz->ygref = thiz->gref.rad;    // otx: setMdlOutScalar
  thiz->yabOsc = thiz->abOsc;    // otx: setMdlOutStructCpy
} // step_OrthBandpassFilter
```

2 Module and test PID-Ctrl with different data types



PIDcontrol1-N.png

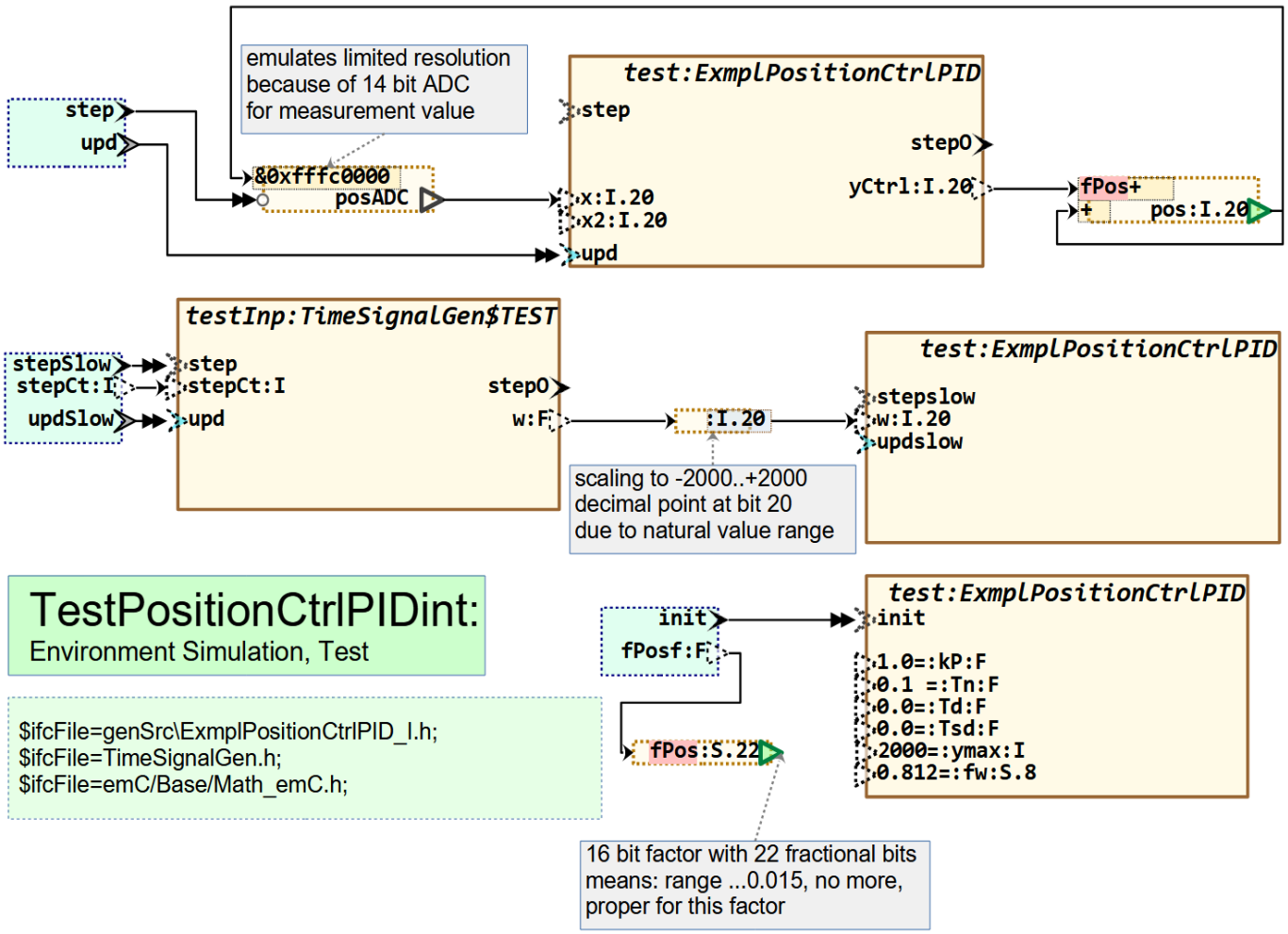
Generated code snippet for Integer arithmetic `ExmplPositionCtrlPID_I.c`:

```
/**Operation step(...)
 */
void step_ExmplPositionCtrlPID_I ( ExmplPositionCtrlPID_I_s* thiz
  , int32 x
  , int32 x2
) {

  thiz->mEvout_step = 0;  // set evout bits to 0, ... otx: GenCode_cObj#mEvout//
  // following statements from exec: prcEvchainOperation(...)

  step_PIDI_Ctrl_emC(&thiz->ctrl, (-x + thiz->w1_z), 0);  //$module_ExmplPosition...
  //
  //Module outputs due to the event stepO:  otx: evMldOutStart
  thiz->mEvout_step |= MASK_step_stepO;  // set the output event bit otx: evMldOutStart
  thiz->yCtrl = (thiz->ctrl.y + mpyIS_emC(thiz->w1_z, thiz->fw_z, 20, 8) );    // otx:...
  //end Module outputs due to the event stepO:  otx: evMldOutEnd
  //

} // step_ExmplPositionCtrlPID_I
```

`mpyIS_emC`: Multiplication int32 * int16 with given number of fractional bits can be implemented by 32 x 32 bit mult, and shift, or by specific asm operations, or by algorithm which tests shift before / shift after with 32 bit multiplications

The module code is generated by knowledge of the calling condition in a specific file names `ExmplPositionCtrlPID_I.c` and `...h` and calls the `PIDI_Ctrl_emC` due to the actual Data type of pin `wx`. The PIDI_Ctrl_emC is given by legacy code

(see https://www.vishia.org/emc/html/Ctrl/PIDctrl.html)



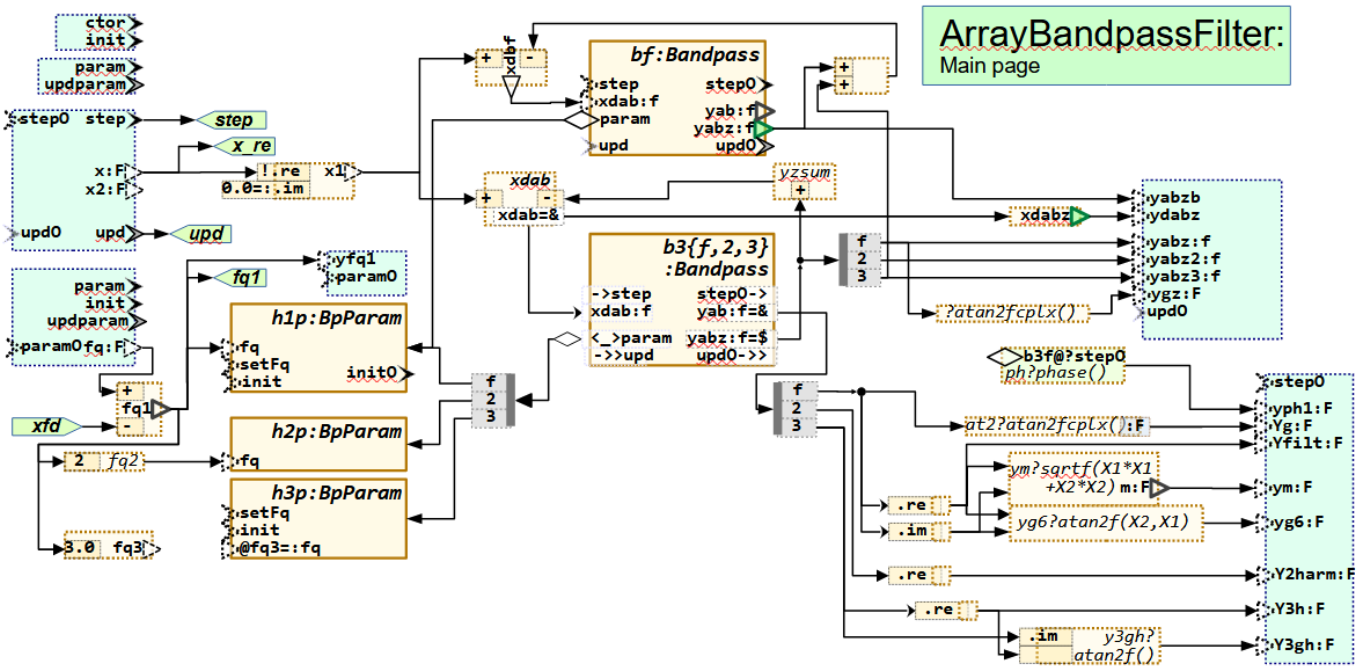
PIDtestEnvironment-I.png

Generated code snippet `TestPositionCtrlPIDint.c`:

```
/**Operation step(...)
 */
void step_TestPositionCtrlPIDint ( TestPositionCtrlPIDint_s* thiz
) {

  thiz->mEvout_step = 0;  // set evout bits to 0, will be se ... otx: GenCode_cObj#mEvout//
  // following statements from exec: prcEvchainOperation(...)


  thiz->posADC = ((thiz->pos_z & 0xfffc0000) );  //step --> posADC_X.prep  genExprOut(...)
  step_ExmplPositionCtrlPID_I(&thiz->test, thiz->posADC, 0);  //posADC_X.prepO --> test.st...

  thiz->pos = (mpyIS_emC(thiz->test.yCtrl, thiz->fPos_z, 20, 22)  + thiz->pos_z);  //...

} // step_TestPositionCtrlPIDint
```

Integer formats `I.12` and `S.13` for `int32` and `int16` with given number of fractional bits for natural values or one byte integer part (`I.24`, `S.8`) for scaling -100..100% with overdrive to 127%.

3 Slices and Arrays



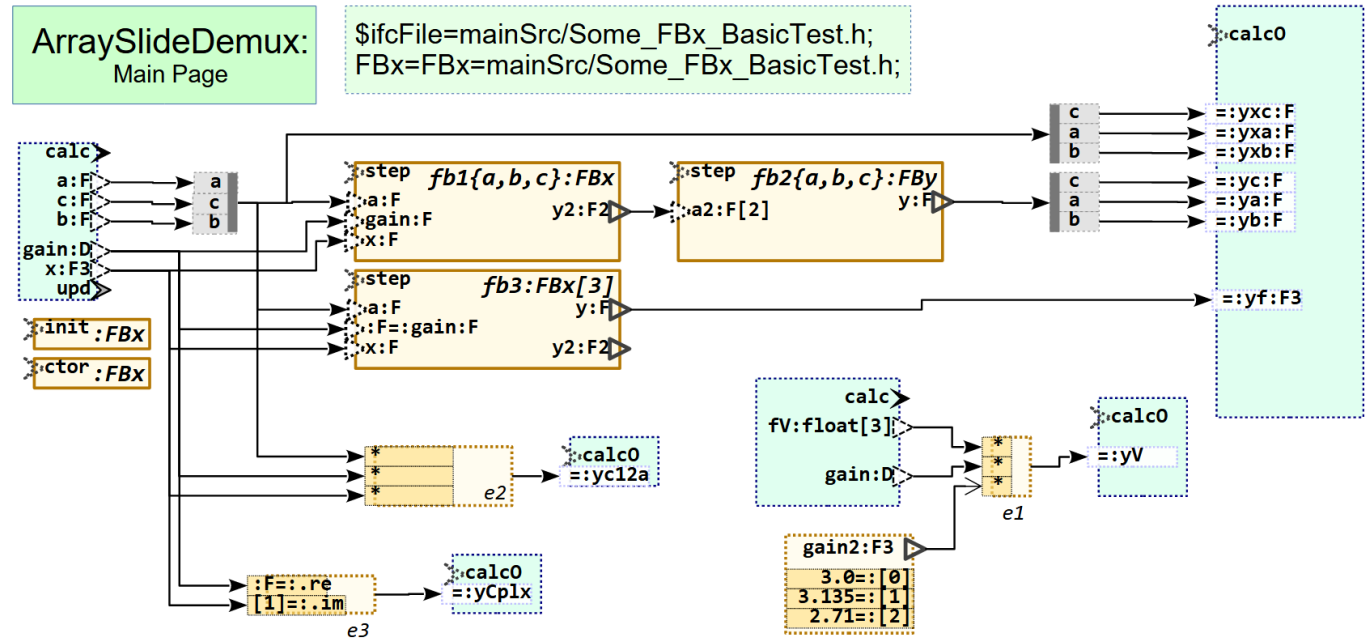
ArrayBandpassFilter-1.odg.png

It's similar The OrthBandpassFilter module but replaces the 3 FBlocks for three harmonics with one GBlock in graphic, which presents 3 FBlock instances. It is "sliced"

Access to non sliced signals are wired with a multiplexer / demux.

The slices have names here, alternative also arrays are possible.

```
bool init_ArrayBandpassFilter ( ArrayBandpassFilter_s* thiz, float fq, float fq) {
  bool bInitOk = true;            // return value for this init_operation
  bool bInitOk1;                  // variable for the init_...() results
  float fq3;  // locale variable for fq3_X.prep otx: defineLocalVar_FBexpr
  bInitOk1 = init_OrthBandpassF_Ctrl_emC(&thiz->b32, &thiz->h2p);        //$module_ArrayBan
  bInitOk &= bInitOk1;   //note: do not combine &= init_...(...) it prevents execution if b
  bInitOk1 = init_OrthBandpassF_Ctrl_emC(&thiz->b33, &thiz->h3p);        //$module_ArrayBan
  bInitOk &= bInitOk1;   //note: do not combine &= init_...(...) it prevents execution if b
  bInitOk1 = init_OrthBandpassF_Ctrl_emC(&thiz->b3f, &thiz->h1p);        //$module_ArrayBan
  bInitOk &= bInitOk1;   //note: do not combine &= init_...(...) it prevents execution if b
  bInitOk1 = init_OrthBandpassF_Ctrl_emC(&thiz->bf, &thiz->h1p);        //$module_ArrayBand
  bInitOk &= bInitOk1;   //note: do not combine &= init_...(...) it prevents execution if b
  thiz->fq1 = (fq - thiz->fd2_z);  //init --> fq1_X.prep  genExprOut(...) in otx: setVar_FB
  ..... }

void step_ArrayBandpassFilter ( ArrayBandpassFilter_s* thiz, float x, float x2) {
  float_complex x1;  // locale variable for x1_X.prep otx: defineLocalVar_FBexpr
  float_complex dq2;  // locale variable for dq2_X.dq otx: defineLocalVar_FBexpr
  float_complex dqref;  // locale variable for dqref_X.dq otx: defineLocalVar_FBexpr
  .....
  thiz->xdab.re = (x1.re - (thiz->b32.yabz.re + thiz->b33.yabz.re + thiz->b3f.yabz.re));
  thiz->xdab.im = (x1.im - (thiz->b32.yabz.im + thiz->b33.yabz.im + thiz->b3f.yabz.im));  /
setVar_FBexpr
  step_OrthBandpassF_Ctrl_emC(&thiz->b32, thiz->xdab);  //xdab_X.prepO --> b32.step  otx:ca
  step_OrthBandpassF_Ctrl_emC(&thiz->b33, thiz->xdab);  //xdab_X.prepO --> b33.step  otx:ca
  step_OrthBandpassF_Ctrl_emC(&thiz->b3f, thiz->xdab);  //xdab_X.prepO --> b3f.step  otx:ca
  ..... }
```



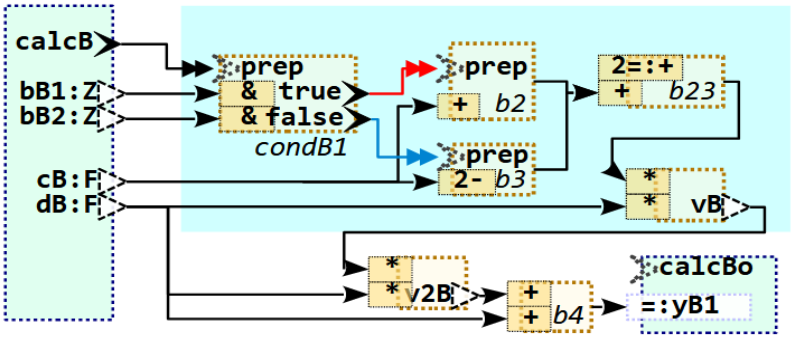
ArraySlideDemux.odg.png

```
void calc_ArraySlideDemux ( ArraySlideDemux_s* thiz
  , float a, float c, float b, double gain, float x[3], float fV[3]) {
  thiz->mEvout_calc = 0;  // set evout bits to 0, will be set maybe conditional otx: GenCo
  // following statements from exec: prcEvchainOperation(...)

  step_FBx(&thiz->fb1a, a, gain, x[0]);  //$module_ArraySlideDemux.calc --> fb1a.step  otx
  step_FBy(&thiz->fb2a, thiz->fb1a.y2);  //fb1a.stepO --> fb2a.step  otx:callFB_evin
  step_FBx(&thiz->fb1b, b, gain, x[1]);  //$module_ArraySlideDemux.calc --> fb1b.step  otx
  step_FBy(&thiz->fb2b, thiz->fb1b.y2);  //fb1b.stepO --> fb2b.step  otx:callFB_evin
  step_FBx(&thiz->fb1c, c, gain, x[2]);  //$module_ArraySlideDemux.calc --> fb1c.step  otx
  step_FBy(&thiz->fb2c, thiz->fb1c.y2);  //fb1c.stepO --> fb2c.step  otx:callFB_evin
  step_FBx(&thiz->fb3[0], a, (float)gain, x[0]);  //$module_ArraySlideDemux.calc --> fb3.s
  step_FBx(&thiz->fb3[1], c, (float)gain, x[1]);  //$module_ArraySlideDemux.calc --> fb3.s
  step_FBx(&thiz->fb3[2], b, (float)gain, x[2]);  //$module_ArraySlideDemux.calc --> fb3.s

  //Module outputs due to the event calcO:  otx: evMldOutStart
  thiz->mEvout_calc |= MASK_calc_calcO;  // set the output event bit otx: evMldOutStart
  thiz->yCplx.im = (x[1]);  //JOIN_calcO.J --> calcO  genExprOut(...) in otx: setVar_FBexp
  thiz->yCplx.re = (((float)gain));  //JOIN_calcO.J --> calcO  genExprOut(...) in otx: set
  thiz->yc12a[0] = (a * gain * x[0]); // otx: setVar_FBexpr dtype.sizeArray >3
  thiz->yc12a[1] = (c * gain * x[1]); // otx: setVar_FBexpr dtype.sizeArray >3
  thiz->yc12a[2] = (b * gain * x[2]); // otx: setVar_FBexpr dtype.sizeArray >3    //JOIN_c

  thiz->yV[0] = (fV[0] * gain * thiz->gain2[0]); // otx: setVar_FBexpr dtype.sizeArray >3
  thiz->yV[1] = (fV[1] * gain * thiz->gain2[1]); // otx: setVar_FBexpr dtype.sizeArray >3
  thiz->yV[2] = (fV[2] * gain * thiz->gain2[2]); // otx: setVar_FBexpr dtype.sizeArray >3

  thiz->yxc = c;    // otx: setMdlOutScalar
  thiz->yxa = a;    // otx: setMdlOutScalar
  thiz->yxb = b;    // otx: setMdlOutScalar

  thiz->yc = thiz->fb2c.y;    // otx: setMdlOutScalar
  thiz->ya = thiz->fb2a.y;    // otx: setMdlOutScalar
  thiz->yb = thiz->fb2b.y;    // otx: setMdlOutScalar
  for(int ix = 0; ix < 3; ++ix) {
    thiz->yf[ix] = thiz->fb3[ix].y;    // otx: setVar_DoutFBarray - scalar dout
  }
} // calc_ArraySlideDemux
```

4 Conditional execution

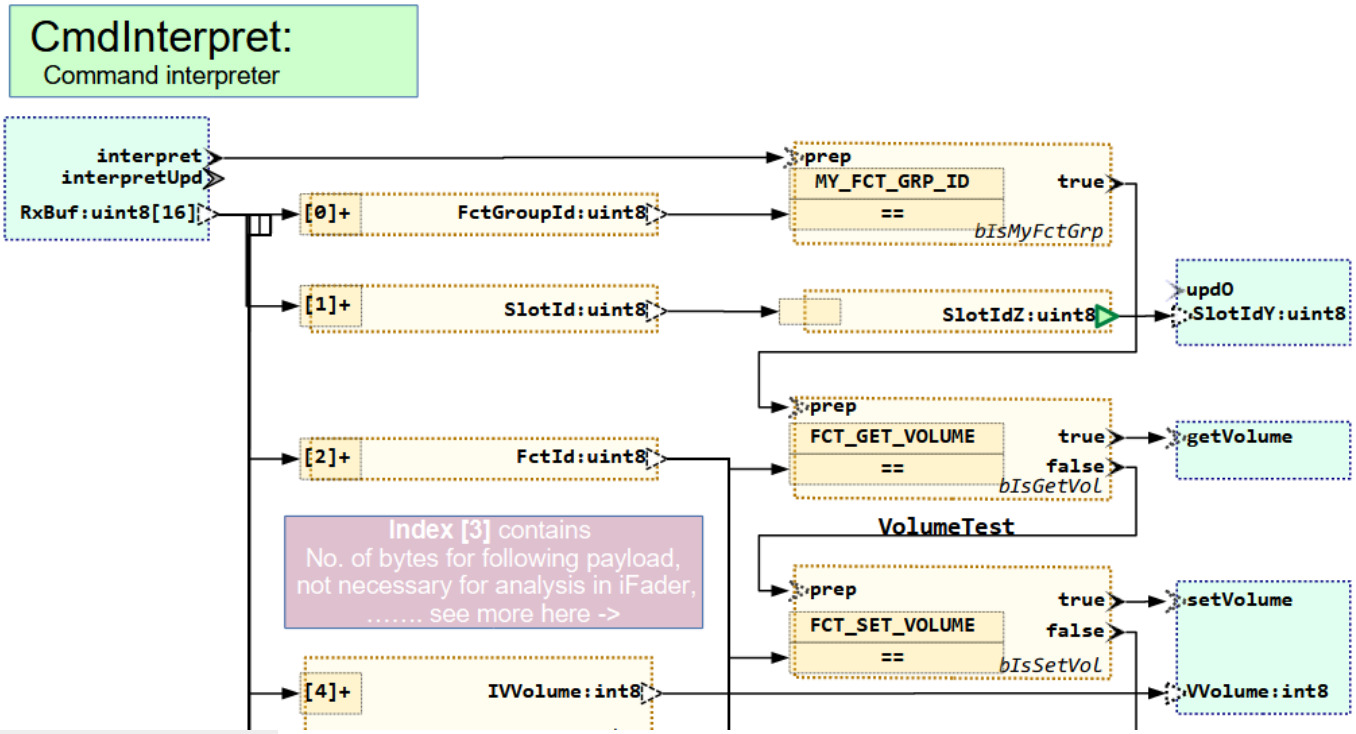
exmpEvTree1.odg.png

The special FBlock with true / false events controls the further evaluation. Either the path starting on b2 or starting on b3 determines the value of `vB`. b23 hat two inputs, both alternatively.




```
/**Operation calcB(...)
 */
void calcB_exmpEvTree1 ( exmpEvTree1_s* thiz
  , bool bB1
  , bool bB2
  , float cB
  , float dB
) {
  float vB;  // locale variable for vB_X.prep otx: defineLocalVar_FBexpr
  float v2B;  // locale variable for v2B_X.prep otx: defineLocalVar_FBexpr

  bool condB1 = false;    // bool for conditions otx: GenCode_cObj#boolCond
  thiz->mEvout_calcB = 0;  // set evout bits to 0, will be set maybe conditional otx:...
  // following statements from exec: prcEvchainOperation(...)

  condB1 = (bB1 && bB2);  // condB1.prep otx: ExprEv_OFB @7'70(59..69, 69..75)
  if( condB1 ) {   // otx: exprCondIf

    vB = ((2 + (cB)) * dB);  //condB1.true --> vB_X.prep  genExpr... in otx: setVar_FBexpr
  } else {   //else (bB1 && bB2)  otx: exprElse

    vB = ((2 + (-(cB * 2) )) * dB);  //condB1.false --> vB_X.prep  ... in otx: setVar_FBexpr
  } // endif   // otx: exprEndif fbx=<null>

  v2B = (vB * dB);  //vB_X.prepO --> v2B_X.prep  genExprOut(...) in otx: setVar_FBexpr
  //
  //Module outputs due to the event calcBo:  otx: evMldOutStart
  thiz->mEvout_calcB |= MASK_calcB_calcBo;  // set the output event bit otx: evMldOutStart
  thiz->yB1 = (v2B + dB);    // otx: setMdlOutScalar
  //end Module outputs due to the event calcBo:  otx: evMldOutEnd
  //

} // calcB_exmpEvTree1
```

The `bool` variable for conditions, instead write the condition immediately in `if(...)`, is used because the event chains may be more complex. The compiler will be optimize the variable for the simple cases.

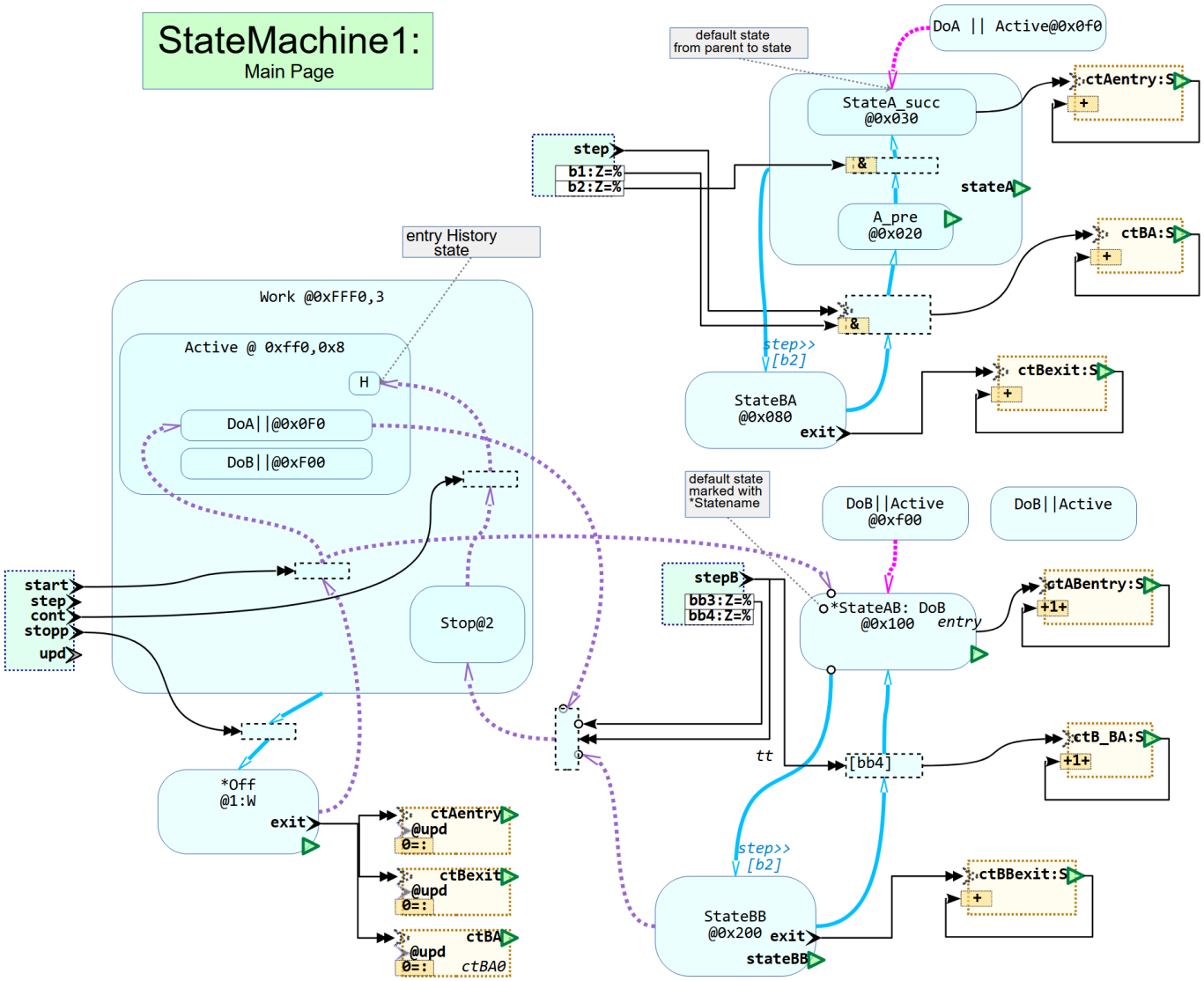


CmdInterpret-1.odg.png

Using example: Evaluation of some commands

```
void interpret_CmdInterpret ( CmdInterpret_s* thiz, uint8 RxBuf[16] ) {
  uint8 FctGroupId;  // locale variable for FctGroupId_X.prep dtype:0`2:V otx: defineLocalV
  uint8 FctId;  // locale variable for FctId_X.prep dtype:0`2:V otx: defineLocalVar_FBexpr
  int8 IIVolume;  // locale variable for IIVolume_X.prep dtype:0`3:B otx: defineLocalVar_FB
  uint8 SlotId;  // locale variable for SlotId_X.prep dtype:0`2:V otx: defineLocalVar_FBexp
  int8 IVVolume;  // locale variable for abcc.prep dtype:0`3:B otx: defineLocalVar_FBexpr

  bool bIsMyFctGrp = false;    // bool for conditions otx: GenCode_cObj#boolCond
  bool bIsGetVol = false;    // bool for conditions otx: GenCode_cObj#boolCond
  bool bIsSetVol = false;    // bool for conditions otx: GenCode_cObj#boolCond
  thiz->mEvout_interpret = 0;  // set evout bits to 0, will be set maybe conditional otx: Ge
  // following statements from exec: prcEvchainOperation(...)
  FctGroupId = (RxBuf[0]);  //interpret --> FctGroupId_X.prep  genExprOut(...) in otx: setVa
  bIsMyFctGrp = (MY_FCT_GRP_ID == FctGroupId) ;  // bIsMyFctGrp.prep otx: ExprEv_OFB @3'90(9
  FctId = (RxBuf[2]);  //interpret --> FctId_X.prep  genExprOut(...) in otx: setVar_FBexpr
  IVVolume = (RxBuf[4]);  //interpret --> abcc.prep  genExprOut(...) in otx: setVar_FBexpr
  if(bIsMyFctGrp) {        // otx: exprCondBitsIf
    bIsGetVol = (FCT_GET_VOLUME == FctId) ;  // bIsGetVol.prep otx: ExprEv_OFB @3'90(90..121
    if( bIsGetVol ) {   // otx: exprCondIf
      //Module outputs due to the event getVolume:  otx: evMldOutStart
      thiz->mEvout_interpret |= MASK_interpret_getVolume;  // set the output event bit otx:
      //end Module outputs due to the event getVolume:  otx: evMldOutEnd
      //
    } else {   //else (FCT_GET_VOLUME == FctId)   otx: exprElse
      bIsSetVol = (FCT_SET_VOLUME == FctId) ;  // bIsSetVol.prep otx: ExprEv_OFB @3'90(90..1
      if( !bIsSetVol ) {   // otx: exprCondIf
        .....
```

5 State machines



StateMachine1.odg.png

* Adequate Harel Statecharts (UML)
* Possible disperse to several pages in the module (of the graphic)
* States can be repeated (on the other pages) as classes in UML-diagrams
* Combination with other FBlocks !
* Note: also FBlocks can be repeated in other pages (identify the same instance)
* Events and conditions can be drawn - from the pins of FBlocks or module pins - or alternatively given textually (as for all pins).

```
void step_StateMachine1 ( StateMachine1_s* thiz, bool b1, bool b2) {

  bool g_20_13_48 = false;    // bool for conditions otx: GenCode_cObj#boolCond
  bool g_20_13_30 = false;    // bool for conditions otx: GenCode_cObj#boolCond
  thiz->mEvout_step = 0;  // set evout bits to 0, will be set maybe conditional otx: GenCo
  // following statements from exec: prcEvchainOperation(...)
  if(g_20_13_48) {        // otx: exprCondBitsIf
    g_20_13_30 = (b2 && thiz->Apre_DoA_Active_Work.d);  // g_20_13_30.prep otx: ExprEv_OFB
    if( g_20_13_30 ) {   // otx: exprCondIf
  leave_State_OFB(&thiz->Apre_DoA_Active_Work);  //g_20_13_30.true --> Apre_DoA_Active_Wor
  set_State_OFB(&thiz->StateA_DoA_Active_Work);  //g_20_13_30.true --> StateA_DoA_Active_W
  upd_State_OFB(&thiz->StateA_DoA_Active_Work);  //g_20_13_30.true --> StateA_DoA_Active_W
    } // endif   // otx: exprEndif fbx=<null>
  }  //Condition Bits
  g_20_13_48 = (thiz->StateBA_DoA_Active_Work.q && b1);  // g_20_13_48.prep otx: ExprEv_OF
  if( g_20_13_48 ) {   // otx: exprCondIf
  set_State_OFB(&thiz->Apre_DoA_Active_Work);  //g_20_13_48.true --> Apre_DoA_Active_Work.
  upd_State_OFB(&thiz->Apre_DoA_Active_Work);  //g_20_13_48.true --> Apre_DoA_Active_Work.
  leave_State_OFB(&thiz->StateBA_DoA_Active_Work);  //g_20_13_48.true --> StateBA_DoA_Acti

  thiz->ctBexit = (thiz->ctBexit_z);  //StateBA_DoA_Active_Work.exit --> ctBexit_X.prep x
  thiz->ctBexit_z = thiz->ctBexit;    // otx: VarZ_OFB_upd

    thiz->ctBA = (thiz->ctBA_z);  //g_20_13_48.true --> ctBA_X.prep  genExprOut(...) in ot
  } // endif   // otx: exprEndif fbx=<null>

} // step_StateMachine1
```

This is the code produced in the moment adequate software version for FBlocks, not prepared for state execution. (It's in development). TODO one operation for the whole state machine, asks the current state and checks the kind of event.

Todo planned in future: also distributed state machines, one state machine in graphic, implemented in different devices, using communication.

Hint for code generation: It is script controlled, with textual templates for different parts of code. The user can adapt the code appearance in some properties.