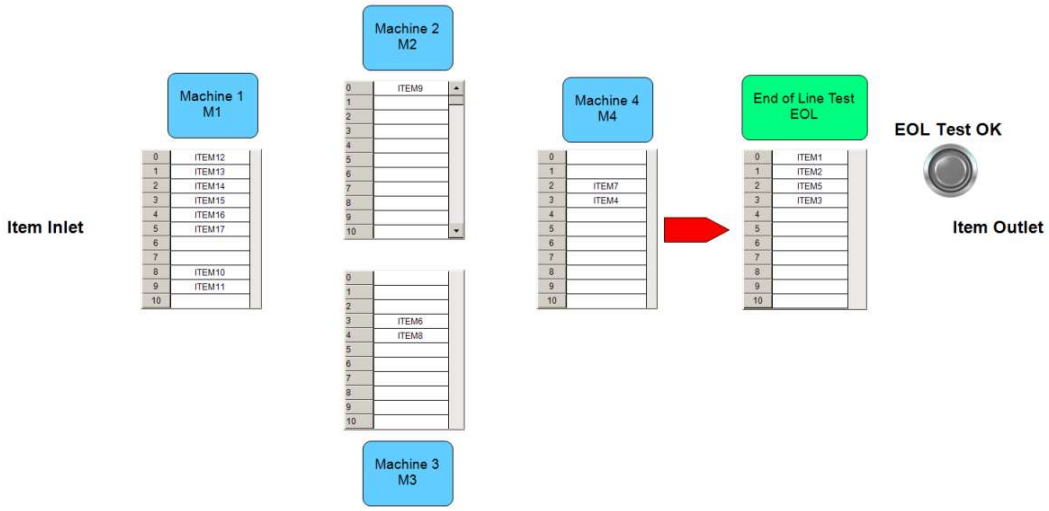
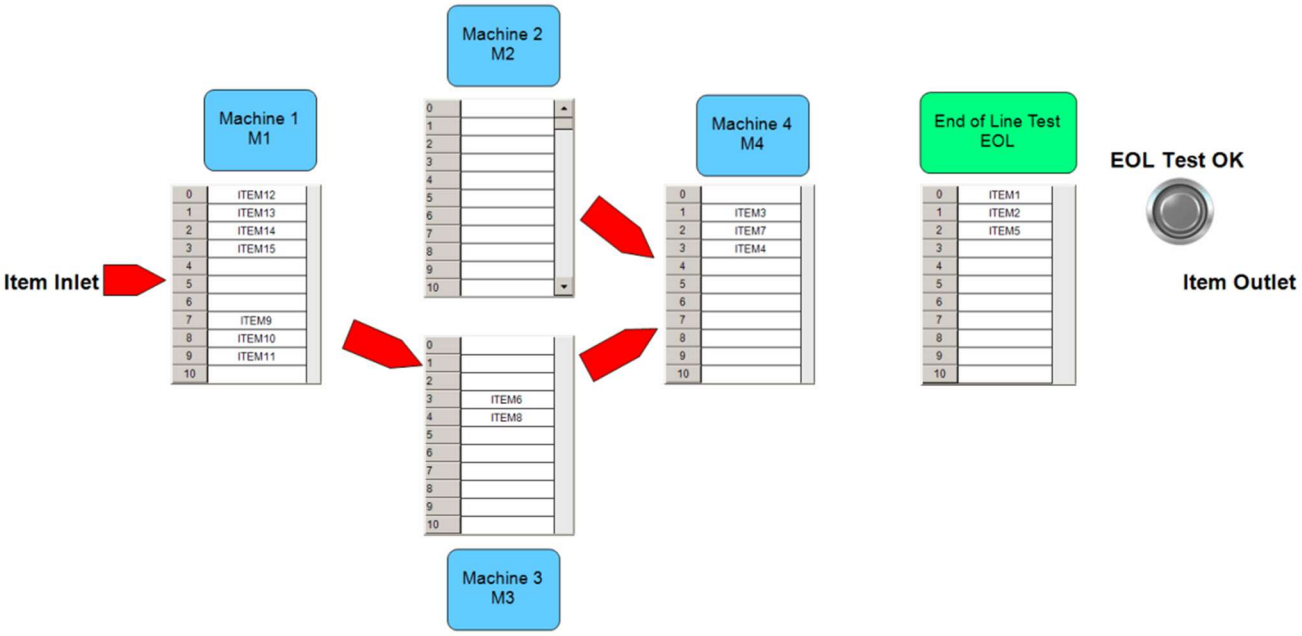
Item distribution with FIFO in factory





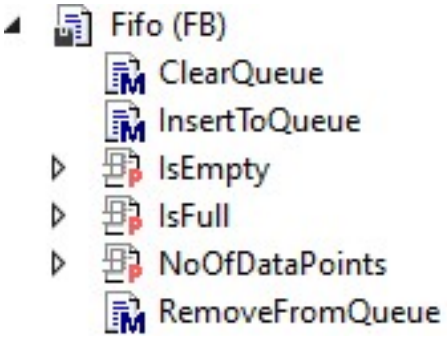
Distribution and processing of items in factory with usage of FIFO circular buffer.
The plant consists of machines 1 to 4 (M1..M4) and End of Line tester station (EOL).
Each machine can process one item at time.

Items are placed at the beginning of the path at item inlet and enter the queue in machine 1.
After one single item is completly processed in M1, it is then diverted into M2 or M3, depending on the amount of free space in respective queue (M2 or M3).
Machine M2 and M3 have different item processing times. Finally, all items are processed in machine 4.
After machine 4 there is also end of line tester with manual button for test confirmation.
Each machine has own buffer (FIFO - circular queue) so that items can wait before entering it.
Each queue can hold maximal of ten items. When an item is placed at the inlet it gets an unique number as *ItemCode*. This number accompanies the item throughout the whole distribution path.

```
PROGRAM MAIN
VAR
	ItemCode		:	STRING(10);
	ItemNo			:	INT;

	tonIn			:	TON;
	ton01			:	TON;
	ton02			:	TON;
	ton03			:	TON;
	ton04			:	TON;

	tpIn			:	TP;
	tpM02In			:	TP;
	tpM03In			:	TP;
	tpM02Out		:	TP;
	tpM03Out		:	TP;
	tpM04Out		:	TP;
	tpEolOut		:	TP;
	Eol			:	BOOL;
	rtrigEol		:	R_TRIG;

	Fifo01			:	Fifo;
	Fifo02			:	Fifo;
	Fifo03			:	Fifo;
	Fifo04			:	Fifo;
	FifoEol			:	Fifo;
END_VAR

// simulation for item inlet
tonIn(IN:=NOT tonIn.Q, PT:=T#1600MS);

// simulation for machine processing M1,M2,M3,M4
ton01(IN:=NOT ton01.Q, PT:=T#3S);
ton02(IN:=NOT ton02.Q, PT:=T#8S);
ton03(IN:=NOT ton03.Q, PT:=T#6S);
ton04(IN:=NOT ton04.Q, PT:=T#7S);

// enf of line test ok
rtrigEol(CLK:=Eol);

Fifo01(QueueLength:=10);   // FIFO for M1
Fifo02(QueueLength:=10);   // FIFO for M2
Fifo03(QueueLength:=10);   // FIFO for M3
Fifo04(QueueLength:=10);   // FIFO for M4
FifoEol(QueueLength:=10);  // FIFO for End of Line station

// timers for visualisation
tpIn(PT:=T#500MS);
tpIn.IN:=FALSE;

tpM02In(PT:=T#500MS);
tpM02In.IN:=FALSE;

tpM03In(PT:=T#500MS);
tpM03In.IN:=FALSE;

tpM02Out(PT:=T#500MS);
tpM02Out.IN:=FALSE;
```

```
tpM03Out(PT:=T#500MS);
tpM03Out.IN:=FALSE;

tpM04Out(PT:=T#500MS);
tpM04Out.IN:=FALSE;

tpEolOut(PT:=T#500MS);
tpEolOut.IN:=FALSE;


// item inlet into M1
IF tonIn.Q AND NOT Fifo01.IsFull THEN
	tpIn.IN := tonIn.Q;
	// move item into M1 if buffer not full
	ItemNo:=ItemNo+1;
	ItemCode:=CONCAT('ITEM',INT_TO_STRING(ItemNo));
	Fifo01.InsertToQueue(DataIn:=ItemCode);
END_IF


// M1 out
IF ton01.Q AND NOT Fifo01.IsEmpty THEN
	IF Fifo02.NoOfDataPoints<Fifo03.NoOfDataPoints AND NOT Fifo02.IsFull THEN
		// move item to M2
		Fifo02.InsertToQueue(DataIn:=Fifo01.RemoveFromQueue());
		tpM02In.IN:=ton01.Q;
	ELSIF Fifo03.NoOfDataPoints<Fifo02.NoOfDataPoints AND NOT Fifo03.IsFull THEN
		// move item to M3
		Fifo03.InsertToQueue(DataIn:=Fifo01.RemoveFromQueue());
		tpM03In.IN:=ton01.Q;
	ELSIF Fifo02.NoOfDataPoints=Fifo03.NoOfDataPoints AND NOT Fifo03.IsFull THEN
		// move item to M3 (M3 has faster processing time)
		Fifo03.InsertToQueue(DataIn:=Fifo01.RemoveFromQueue());
		tpM03In.IN:=ton01.Q;
	END_IF
END_IF

// M2 out
IF ton02.Q AND NOT Fifo02.IsEmpty THEN
	IF NOT Fifo04.IsFull THEN
		Fifo04.InsertToQueue(DataIn:=Fifo02.RemoveFromQueue());
		tpM02Out.IN:=ton02.Q;
	END_IF
END_IF


// M3 out
IF ton03.Q AND NOT Fifo03.IsEmpty THEN
	IF NOT Fifo04.IsFull THEN
		Fifo04.InsertToQueue(DataIn:=Fifo03.RemoveFromQueue());
		tpM03Out.IN:=ton03.Q;
	END_IF
END_IF
```

```
// M4 out
IF ton04.Q AND NOT Fifo04.IsEmpty THEN
	IF NOT FifoEol.IsFull THEN
		FifoEol.InsertToQueue(DataIn:=Fifo04.RemoveFromQueue());
		tpM04Out.IN:=ton04.Q;
	END_IF
END_IF

// M5 out
IF rtrigEol.Q AND NOT FifoEol.IsEmpty THEN
	FifoEol.RemoveFromQueue();
	tpEolOut.IN:=rtrigEol.Q;
END_IF
```

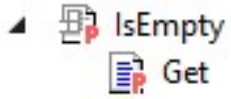



```
FUNCTION_BLOCK Fifo
VAR_INPUT
	QueueLength		:	INT;
END_VAR

VAR_OUTPUT
	ErrorMsg		:	STRING(50);	// error message
	QueueFull		:	Bool;
	QueueEmpty		:	BOOL;
END_VAR

VAR CONSTANT
	QueueMin		:	INT:=0;
	QueueMax		:	INT:=29;
END_VAR

VAR
	ptrHead			:	INT ;		// pointer to first element (head)
	ptrTail			:	INT ;		// pointer to last element (tail)
	Idx			:	INT;
	InitBit			:	BOOL:=TRUE;
	QueueSize		:	INT;		// size of circular queue
	QueueSizeOk		:	BOOL;
	Queue			:	ARRAY[QueueMin..QueueMax] OF STRING(10);
END_VAR
```

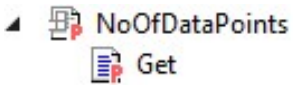
```
(*
This FB acts as FIFO i.e. circular queue.
Size of the queue must be customized with constants QueueMin and QueueMax.
*)
IF InitBit THEN

	InitBit := FALSE;

	IF QueueMax > QueueMin AND QueueLength>1 THEN
		ErrorMsg:='';
		ptrHead := -1;
		ptrTail := -1;
		QueueSize := TO_INT(SIZEOF(Queue) / SIZEOF(REAL));
		QueueSizeOk := TRUE;
	ELSE
		QueueSizeOk:=FALSE;
		ErrorMsg:='Queue Length must be > 1';
	END_IF
END_IF



METHOD InsertToQueue : BOOL
VAR_INPUT
	DataIn	:	STRING(10);		// data to insert into circular queue
END_VAR

// Insert new data into queue

IF QueueSizeOk THEN
	IF IsFull THEN
		// queue is full. can't insert new data
		ErrorMsg := 'Queue is full. Can not insert new data';
		QueueFull := TRUE;
	ELSIF IsEmpty THEN
		ErrorMsg := '';
		QueueFull := FALSE;
		ptrHead := 0;
		ptrTail := 0;
		Queue[ptrTail] := DataIn;
	ELSE
		ErrorMsg := '';
		QueueFull := FALSE;
		IF (ptrTail+1) MOD QueueLength <= QueueLength THEN
			ptrTail := (ptrTail + 1) MOD QueueLength;
			Queue[ptrTail] := DataIn;
		END_IF
	END_IF
END_IF
```

```
METHOD RemoveFromQueue : STRING(10);
VAR_INPUT
END_VAR

// remove one data from the queue

IF QueueSizeOk THEN
	IF IsEmpty THEN 					// zero element in queue
		ErrorMsg := 'Queue is empty';
		QueueEmpty := TRUE;
	ELSIF ptrHead=ptrTail THEN 			// only one element in queue
		ErrorMsg := '';
		QueueEmpty := FALSE;
		RemoveFromQueue := Queue[ptrHead];
		Queue[ptrHead]:='';
		ptrHead := -1;
		ptrTail := -1;
	ELSE 						// more then one element in queue
		ErrorMsg := '';
		QueueEmpty := FALSE;
		IF (ptrHead + 1) MOD QueueLength <= QueueLength THEN
			RemoveFromQueue := Queue[ptrHead];
			Queue[ptrHead]:='';
			ptrHead := (ptrHead + 1) MOD QueueLength;
		END_IF
	END_IF
END_IF



METHOD ClearQueue : BOOL
VAR_INPUT
END_VAR

// reinit the queue
IF QueueSizeOk THEN
	ptrHead := -1;
	ptrTail := -1;

	FOR Idx := QueueMin TO QueueMax DO
		Queue[Idx] := '';
	END_FOR

	ErrorMsg := '';
	QueueEmpty := FALSE;
	QueueFull := FALSE;
END_IF
```


```
PROPERTY IsFull : BOOL

Fifo.IsFull.Get
IsFull := ptrHead = (ptrTail + 1) MOD QueueLength;
```




```
PROPERTY IsEmpty : BOOL

Fifo.IsEmpty.Get
IsEmpty := ptrHead = -1 AND ptrTail = -1;
```




```
PROPERTY NoOfDataPoints : INT

Fifo.NoOfDataPointes.Get
// number of data points in queue
IF ptrTail>ptrHead THEN
      NoOfDataPoints:=ABS(ptrTail - ptrHead)+1;
ELSIF ptrTail<ptrHead THEN
      NoOfDataPoints:=(ptrTail+1) + (QueueLength-ptrHead);
ELSE
      // ptrTail=ptrHead
      NoOfDataPoints:=0;
END_IF
```