**Subject:** Re: ANATAC Meeting, July 7th
**From:** Barry Clark <bclark@aoc.nrao.edu>
**Date:** Tue, 6 Jul 2004 14:38:00 -0600 (MDT)
**To:** demerson@nrao.edu, ldaddario@nrao.edu
**CC:** pnapier@zia.aoc.nrao.edu, bclark@nrao.edu, jpayne@nrao.edu, athompso@nrao.edu, rsramek@nrao.edu, ldaddari@nrao.edu, asymmes@nrao.edu

Thinking about the theory of Walsh functions was making my head hurt,
so I quit thinking and just wandered around looking for a suitable set.
I found a set of length 256 (for the correlator; as Larry points out,
the 180d reversals can come faster and be removed at the sampler) that
works for 64 antennas.  I attach a program that demonstrates them.  The
180d phase reversal sequence is 512 long.  I hypothesize that similar
sequences can be found with faster phase reversal sequencies, sufficient
to suppress problems with any timescale, up to the point that it starts
to bother the locking of the first LO.  (There's a lot more stuff to try
for the higher sequencies, so it should be easier to find something that
works than in the 256/512 case.)

I'm not sure there isn't a 128 step sequence, I just didn't happen to
stumble across one that worked.  Even with that small dataset, one
can't search exhaustively, and I couldn't come up with any theoretical
guidance to narrow the search.

The full nested Walsh implementation has the advantage that the 180d
transitions need not be synced with the 90d ones - they remain orthagonal
even if slipped.

---

```
#define N 512
#define LN 9
#define NA 256
#define LNA 8
#define DEL 1
main()
```

```
{
int i, j, k, l, m;
int sr, si, srt, sit, gi, git, qit;
short wf[N] [N];
short af[NA] [NA];

/* make Walsh functions */
af[NA-2][NA-2] = af[NA-2][NA-1] = af[NA-1][NA-2] = 1;
af[NA-1][NA-1] = -1;
j=1;
for (i=0; i< LNA-1; i++)
    {
    j *= 2;
    for (k=0; k<j; k++) for(l=0; l<j; l++)
        {
        af[NA-2*j+k][NA-2*j+l] = af[NA-j+k][NA-j+l];
        af[NA-  j+k][NA-2*j+l] = af[NA-j+k][NA-j+l];
        af[NA-2*j+k][NA-  j+l] = af[NA-j+k][NA-j+l];
        af[NA-  j+k][NA-  j+l] = -af[NA-j+k][NA-j+l];
        }
    }

/* now we double the size of the af[][] rows and use for the
   90d pattern, and then fake something more exciting for the
   180d pattern */
for(i=0; i<NA; i++)
    {
    k = l = 0;
    for(j=0; j<NA; j++)
        {
        wf[2*i][2*j] = wf[2*i][2*j+1] = af[i][j];
        if(af[i][j]==1)
            {
            wf[2*i+1][2*j] = af[i][k%64];
            wf[2*i+1][2*j+1] = af[i][(k++)%64];
            }
        else
            {
```

```
            wf[2*i+1][2*j] = af[i][l%64];
            wf[2*i+1][2*j+1] = af[i][(l++)%64];
            }
        }
    }

m = N/NA;

/* check all baselines */
qit = git = srt = sit = 0;
for(i=0; i<NA; i++)
    {
    gi = 0;
    for (j=i+1; j<NA; j++)
        {
        sr = si = 0;
        for(k=0; k< N; k++)
            {
            if (wf[m*i][k] * wf[m*j][k] == 1)
               sr += wf[m*i+DEL][k] * wf[m*j+DEL][k];
            else if (wf[m*i][k] == 1)
               si += wf[m*i+DEL][k] * wf[m*j+DEL][k];
            else
               si -= wf[m*i+DEL][k] * wf[m*j+DEL][k];
            }
        if(sr!=0) srt++;
        if(si!=0) sit++;
        if(si!=0 || sr!=0)
            {
            printf("(%d,%d)  ", i, j);
            gi++;
            }
        }
    if(gi==0)
       git++;
    if(gi==0 && (i%4)==0)
       qit++;
    }
```

```
printf("\ntotal failures: %d real, %d imag\n", srt, sit);
printf("Total antennas supported: %d\n", git);
printf("Total antennas supported with quarter rate on 90d: %d\n",qit);

}
```